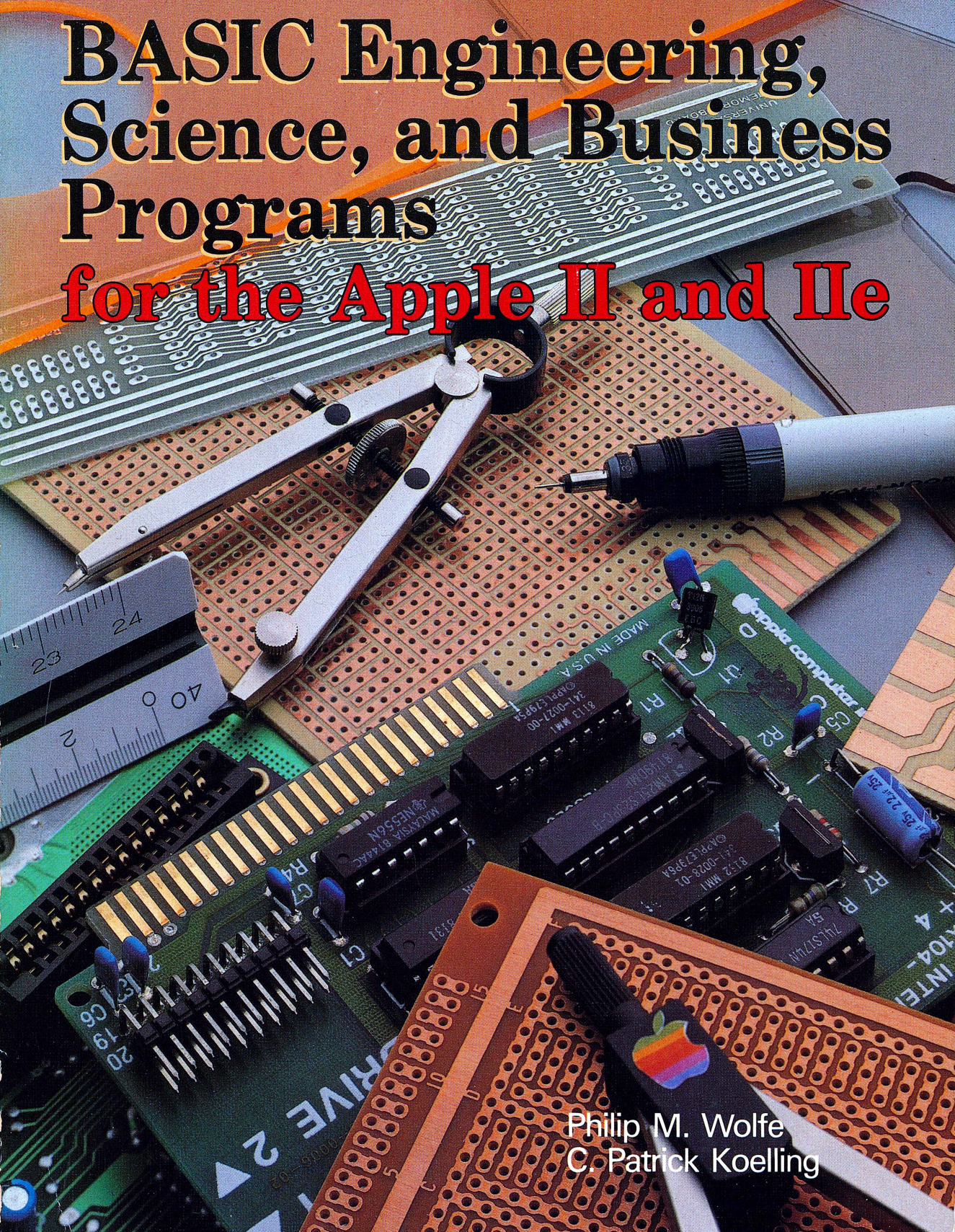
BASIC Engineering, Science, and Business Programs

for the Apple II and IIe



Philip M. Wolfe
C. Patrick Koelling

BASIC Engineering, Science and Business Programs for the Apple II and IIe

Publishing Director: David Culverwell
Acquisitions Editor: Terrell Anderson
Production Editor: Len Blasso
Text Design: Michael Rogers
Art Director/Cover Design: Don Sellers
Assistant Art Director: Bernard Vervin
Cover Photo: George Dodson
Manufacturing Director: John A. Komsa
Indexer: William O. Lively
Typesetter: Harper Graphics, Waldorf, MD
Printer: Fairfield Graphics, Fairfield, PA
Typefaces: Times Roman (text), Helvetica (display), Computer (programs)

BASIC Engineering, Science, and Business Programs for the Apple II and IIe

Philip M. Wolfe
C. Patrick Koelling

Brady Communications Co., Inc., Bowie, MD 20715
A Prentice-Hall Publishing Company

Note to Authors

Do you have a manuscript or a software program related to personal computers? Do you have an idea for developing such a project? If so, we would like to hear from you. The Brady Co. produces a complete range of books and applications software for the personal computer market. We invite you to write to David Culverwell, Editor-in-Chief, Brady Communications Co., Inc., Bowie, Maryland 20715.

BASIC Engineering, Science and Business Programs for the Apple II and IIe

Copyright © 1984 by Brady Communications Co., Inc.
All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, or by any information storage and retrieval system, without permission in writing from the publisher. For information, address Brady Communications Co., Bowie, Maryland 20715

Library of Congress Cataloging in Publication Data

Wolfe, Philip.
BASIC engineering, science, and business programs for the Apple II and IIe.
Includes index.
1. Apple II (Computer)—Programming. 2. Apple IIe (Computer)—Programming. 3. Basic (Computer program language) 4. Computer programs. I. Koelling, C. Patrick, 1953– . II. Title. III. Title: B.A.S.I.C. engineering, science, and business programs for the Apple II and IIe.
QA76.8.A662W65 1984 001.64'25 84-6412
ISBN 0-89303-284-0

Prentice-Hall International, Inc., London
Prentice-Hall Canada, Inc., Scarborough, Ontario
Prentice-Hall of Australia, Pty., Ltd., Sydney
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Prentice-Hall of Southeast Asia Pte. Ltd., Singapore
Whitehall Books, Limited, Petone, New Zealand
Editora Prentice-Hall Do Brasil LTDA., Rio de Janeiro

Printed in the United States of America

84 85 86 87 88 89 90 91 92 93 94 1 2 3 4 5 6 7 8 9 10

TO OUR PARENTS,

EARL AND HELEN WOLFE
CHARLES AND BETTIE KOELLING

CONTENTS

Acknowledgments

Preface

Dedication

1 **The Apple IIe** 1
1.1 Significant Features 1
1.2 Types of Applications 3
1.3 The BASIC Interpreter 4
1.4 Summary 9
References 9
Exercises 9

2 **Data Reduction** 11
2.1 Basics of Data Reduction 11
2.2 Measures of Central Tendency 12
2.3 Measures of Dispersion 15
2.4 Data Plots 18
2.5 Data Analysis and Plotting Subroutines 18
2.6 Summary 33
References 33
Exercises 33

3 **Matrices & Vectors** 35
3.1 Definitions of a Matrix 35
3.2 Matrix Operations 37
3.3 Vectors 49
3.4 Summary 49
References 49
Exercises 50

4 **Curve Fitting with Linear Regression** 51
4.1 Linear Regression 51
4.2 Multiple Regression 57
4.3 Stepwise Regression 58
4.4 Summary 104
References 104
Exercises 104

5 **Solution of Simultaneous Linear Equations** 107
5.1 Simultaneous Linear Equations-General Form 107
5.2 Matrix Inverse Method 108
5.3 Gauss-Jordan Method 109
5.4 Gauss-Seidel Method 114
5.5 Summary 121
References 121
Exercises 121

6 **Roots of Polynomials** 123
6.1 Roots of Polynomials 123
6.2 Newton-Raphson Method 124
6.3 Summary 135
References 135
Exercises 135

7 **Numerical Integration** 137
7.1 Basics of Integration 137
7.2 Numerical Methods 138
7.3 Summary 143
References 143
Exercises 144

8 **Numerical Solutions to Differential Equations** 145
8.1 Overview of Differential Equations 145
8.2 Numerical Solutions 147
8.3 Application of the Runge-Kutta Method 149
8.4 Summary 152
References 154
Exercises 154

9 **Linear Programming** 157
9.1 Linear Programming 157
9.2 Basic Formulation 158
9.3 Solution Methods 163
9.4 Using the Simplex Program 166
9.5 Summary 178
References 178
Exercises 178

10 **Forecasting with Exponential Smoothing** 181
10.1 Single Exponential Smoothing 181
10.2 Exponential Smoothing with a Trend 183
10.3 Seasonal Exponential Smoothing 183
10.4 Exponential Smoothing Program 186

10.5 Summary 210
References 210
Exercises 210

11 Project Planning & Scheduling with CPM 213
11.1 The CPM Technique 213
11.2 CPM Program 215
11.3 Example CPM Problem 222
11.4 Summary 229
References 230
Exercises 230

12 Sorting 233
12.1 Bubble Sort 233
12.2 Quicksort 235
12.3 Summary 241
References 241
Exercises 241

13 Disk Data Files 243
13.1 Recording Information on a Diskette 243
13.2 Sequential Access Files 245
13.3 Random Access Files 254
13.4 Summary 267
References 268
Exercises 268

14 Data Structures 269
14.1 Arrays 269
14.2 Stacks 270
14.3 Queues 272
14.4 Linked Lists 274
14.5 Summary 284
References 285
Exercises 286

15 Random Numbers & Simulation 287
15.1 Using the Function RND 287
15.2 Developing a Random Number Generator 289
15.3 Testing a Random Number Series 290
15.4 Random Number Generators for Specific Distributions 294
15.5 Next Event Simulation 300
15.6 Summary 305
References 305
Exercises 306

Index 323

Optional Diskette Documentation 326

PREFACE

Microcomputers are being used in many types of engineering, science, and business applications. Students are learning how to apply microcomputers. However, for many applications on the job or in the classroom a computer may not be used because the appropriate programs are not available. We have provided several programs, in this text, which are often used in engineering, science, and business. These programs can be used by a practitioner or a student.

Microcomputer users have widely varying computer backgrounds. In writing this book, we assumed that the typical reader had the following characteristics:

1. One semester of computer programming or has read an introductory book on BASIC programming.
2. Some computer experience at running BASIC or FORTRAN programs on a microcomputer or on a large computer.
3. No experience writing programs that use disk files and/or data structures.
4. Completed at least college algebra (however, understanding a few of these programs requires some calculus, statistics, and probability theory).

It is impossible, in one book, to consider all of the areas in which a microcomputer might be applied. However, based upon our industrial, research, and teaching experiences, we have compiled a list of the more useful computer applications that are not readily available. In addition, the subjects of disk data files (sequential and random access) and data structures are discussed. These subjects usually are not taught in beginning computer application courses. A fundamental understanding of these topics is required, however, if more complicated computer programs are to be developed.

To run these programs, you will need an Apple II Plus, with 48K bytes of RAM, or IIe, with 64K bytes of RAM, and one disk drive. Our disk operating system was DOS 3.3. All of the programs can be run with a 40-character per line display; some of the programs also have an option which will let you display the output at 80-characters per line. The programs were written using Applesoft. A few of the programs are several pages in length; consequently, entering these programs by hand can be tedious. One solution is to purchase the optional diskette (available from the publisher, a book store, or your computer store) which contains the programs in this book. On the diskette are 23 files containing 38 major programs and several supporting subroutines.

Each chapter in this book addresses a specific technique or group of related techniques. The chapters are written so that most of the material necessary to use a technique or associated computer program is contained within that chapter. Therefore, with a few exceptions, an individual scientist can pick up this book and only refer to the chapter dealing with the specific technique with which he or she is currently concerned. The same approach can be used if this book is being used as a text book in a course. However,

for those who will be reading the book from beginning to end, the material has been arranged in a logical sequence.

Each chapter begins with a brief explanation of the technique being addressed. Next, a program that applies this technique is presented, followed by an example illustrating the use of the program. Each program provided contains remark statements to assist in following the logic. When you add these programs to your library, you should test each one using the data supplied in the examples. In a few instances, we felt that the technique being presented contained theoretical development beyond the scope of this book. In these cases and in all chapters, references are provided for in-depth reading.

Many of the programs have been written as subroutines so that you can integrate them into any program. All line numbers are unique so that any combination of the subroutines can be in memory at one time—provided you have enough memory. Some of the programs are not written as subroutines because we felt that they would not normally be used as such. Some of these stand-alone programs are the exponential smoothing forecasting program, the critical path method (CPM) project scheduling program, and the linear programming program. However, none of these stand-alone programs has line numbers that will conflict with the subroutines line numbers.

Chapter 1, **The Apple IIe**, discusses the unique features of the Apple IIe. Considerations for applying the IIe are presented and the Applesoft interpreter is introduced.

Chapter 2, **Data Reduction**, describes techniques for the evaluation of data. Subroutines are included that measure the central tendency and dispersion of data. Subroutines are also provided to plot histograms and curves.

Chapter 3, **Matrices and Vectors**, defines a vector and a matrix. The most common vector and matrix operations are summarized, including addition and subtraction, transposition, multiplication, and exponentiation. Calculating the determinant of a matrix and inverting a matrix are also discussed. Subroutines are provided for each of these operations.

Chapter 4, **Curve Fitting with Linear Regression**, explains how linear regression can be used to fit a line through a set of data. A program is provided that performs simple linear regression. Some transforms are presented to permit the use of linear regression when the model is non-linear. A second program is provided for multiple and stepwise regression. This program can also weight and/or transform the data.

Chapter 5, **Solution of Simultaneous Linear Equations**, presents algorithms for the Gauss-Jordan and Gauss-Seidel methods of solving simultaneous linear equations. Subroutines are provided for each of these algorithms.

Chapter 6, **Roots of Polynomials**, discusses the numerical solution of such equations. The Newton-Raphson procedure is introduced. A subprogram is provided for this popular technique.

Chapter 7, **Numerical Integration**, presents two techniques for numerically integrating a single-variable function. The chapter begins by reviewing integration concepts. Subprograms are included for the trapezoidal rule and Legendre-Gauss quadrature.

Chapter 8, **Numerical Solutions to Differential Equations**, presents another numerical technique, the powerful Runge-Kutta method, for the solution of differential equations. This examination is restricted to first-order ordinary differential equations.

Chapter 9, **Linear Programming**, deals with the simplex algorithm. Using this procedure, you can optimize a linear objective function constrained by a set of linear inequalities. The chapter begins with a general discussion of the linear programming problem. The algorithm is briefly explained and a linear programming program is presented.

Chapter 10, **Forecasting with Exponential Smoothing**, explains how exponential smoothing can be used for time series forecasting. Some advantages of exponential smoothing are presented. A forecasting algorithm called Winter's method is discussed. This technique can be used when the time series contains complicating factors, such as trend and seasonal components. A program is provided that applies Winter's method.

Chapter 11, **Project Scheduling with CPM**, deals with using the critical path method (CPM) for planning and scheduling a project. This technique is explained using an example project. The same project is also used to illustrate the CPM program that is provided.

Chapter 12, **Sorting**, presents two sorting algorithms: the bubble sort, and the quicksort. Subroutines are provided which apply these algorithms.

Chapter 13, **Disk Data Files**, addresses input and output using disk files. Sequential and random access disk files are discussed. Programs are provided that illustrate the use of both techniques to manage data.

Chapter 14, **Data Structures**, describes three types of structures commonly used to handle data: a stack, a queue, and linked lists. Subprograms are provided for each technique.

Chapter 15, **Random Numbers and Simulation**, first discusses techniques for generating random numbers using the RND function. Techniques are then presented for generating random numbers from the uniform, normal, exponential, Poisson, and Erlang distributions. Subroutines are presented for each of these random number generators. In addition, digital simulation is briefly explained using a program that simulates the operation of a single-server queue.

Tables P.1 and P.2 contain a list of the major subroutines and programs provided in this text. Many other example programs and minor subroutines appear in the text. A diskette containing these major subroutines and programs is available from the publisher.

This book covers a broad range of topics. The intent is to maximize the usefulness of this text to the individuals using a microcomputer at work, and to the students who are just learning to apply the techniques being presented here.

Line Numbers	Discussed in Chapter	File Name	Subroutine Description
7500-12819	2	ANPLOT	Data reduction and plotting
13000-14880	3	MATOP	Matrix operations
18000-20570	5	SIMLQ	Gauss-Jordan and Gauss-Seidel method for solving simultaneous linear equations
20600-20770	6	NEWRAP	Newton-Raphson method for obtaining roots of a polynomial
20900-21140	7	NUMINT	Trapezoidal rule and Legendre-Gauss quadrature for numerical integration
21200-22760	8	DIFEQ	Runge-Kutta and revised Runge-Kutta for numerical solutions to differential equations
23000-23150	12	BUBSORT	Bubble sort
23200-24230	12	QUIKSORT	Quicksort
24500-24780	14	STACK	Stack data structure
24800-25120	14	QUEUE	Queue data structure
25200-25770	14	LLQUEUE	Linked lists where each list is a queue
25800-26870	14	BIDILIST	Bidirectional linked lists where each list is a queue
27000-27160	15	ORGRND	Random number generator
27200-27580	15	CHISQ	Chi-square goodness of fit test
28000-28610	15	DISTGEN	Random number generators for specific distributions

Table P.1—Major Subroutines Provided

Line Numbers	Discussed in Chapter	File Name	Program Description
1000-1990	4	SIMREG	Simple linear regression
1000-8240	4	STEPWISE	Multiple and stepwise linear regression
1000-4000	9	SIMPLEX	Uses the simplex procedure to solve linear programming problems
1000-7640	10	EXSMOOTH	Winter's method for time series forecasting
1000-4460	11	CPM	Critical path method for project management
1000-2490	13	SEQMAN	Illustrates using the sequential access method to manage data
1000-3550	13	RANMAN	Illustrates using the random access method to manage data
1000-2250	15	QUEUESIM	Illustrates next event simulation of a single-server queue

Table P.2—Major Programs Provided

ACKNOWLEDGEMENTS

The creation of any book requires the contributions of many people (our teachers, colleagues and families). We sincerely thank each of you. In particular, we acknowledge the inspiration and assistance provided by the staff of the Brady Company: Charlie Siegel, Terry Anderson, Len Blasso and the many others involved. Good suggestions were provided by the reviewers, Lance Leventhal, Lew Cox, and Lois Graff. Also, Morteza Abtahi spent many hours validating the programs.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The authors and publisher of this book have used their best efforts in preparing this book and the programs contained in it. These efforts include the development, research, and testing of the programs to determine their effectiveness. The authors and the publisher make no warranty of any kind, expressed or implied, with regard to these programs, the text, or the documentation contained in this book. The authors and the publisher shall not be liable in any event for claims of incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of the text or the programs. The programs contained in this book and on any diskettes are intended for use of the original purchaser-user. The diskettes may be copied by the original purchaser-user for backup purposes without requiring express permission of the copyright holder.

1

The Apple IIe

Your Apple Computer is becoming common at the work place of many engineers and scientists. Apple computer sales constitute a large share of the microcomputer market. The programs presented in this text will run on an Apple II-Plus computer or the recently released Apple IIe (e is for ''enhanced'') computer. From this point on, we will refer to the ''Apple Computer'' to mean both the II-Plus and the IIe. A large number of software developers have developed programs especially for the Apple Computer. The availability of a large variety of software enhances the popularity of the Apple Computer and assures it a healthy share of the microcomputer market.

1.1 Significant Features

The Apple Computer is popular because of its numerous useful features. Several of these features will be described in this section. For example, the Apple has an attractive appearance (the IIe is shown in Figure 1.1). It is packaged in one convenient unit. Several types of display devices can be connected to the unit, such as a monochrome and/or color monitor(s), and a dot matrix or letter quality printer.

The IIe will display 96 different characters, with upper and lower case letters and special characters. The IIe is capable of sending all 128 ASCII characters (some are not displayed, such as control characters). The II-Plus does not have the lower case capability, and seven fewer special characters are available. The standard display is 24 lines in a 40-column wide format. The IIe includes 80-column capability, while this requires an extra board in the II-Plus.

The graphics capabilities of the Apple are very good, with both low and high resolution graphics provided. The graphics hardware will support a monochrome or a color display. The two resolutions available provide low resolution (40 horizontal points by 40 vertical points with 16 possible colors) and high resolution (280 by 192 points with 6 possible colors).

The system unit is well desiged (see Figure 1.2). The system board (the principal circuit board) fits horizontally into the bottom. Visible in Figure 1.2 are 7 expansion slots (for extra memory, etc., the II-Plus has 8), power supply, memory (RAM and ROM), speaker, and the 6502 microprocessor and its support elements.

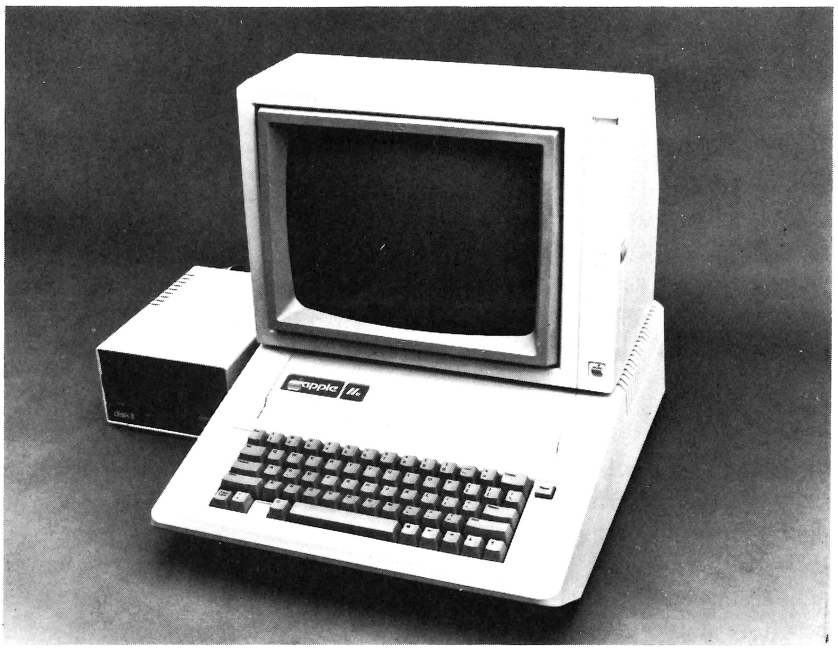


Figure 1.1—Apple IIe Computer

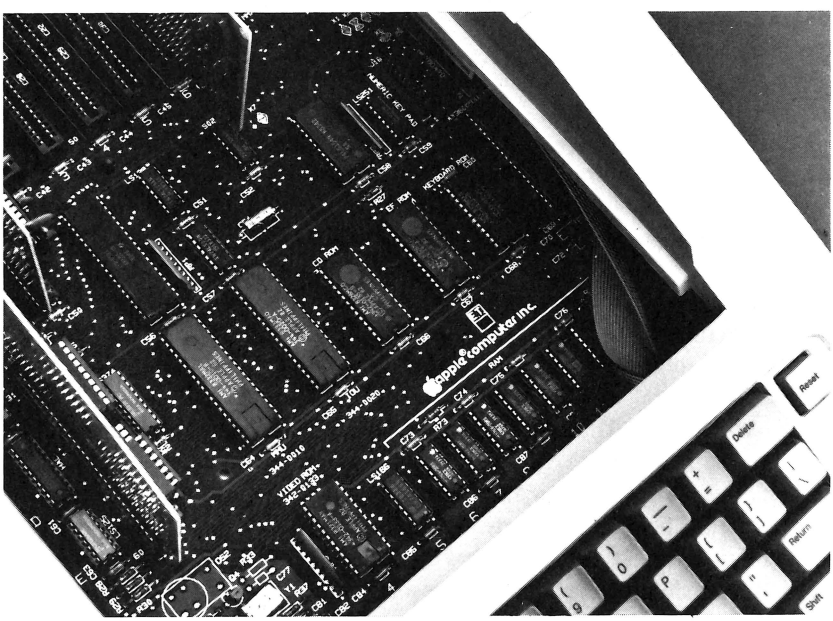


Figure 1.2—Inside the System Unit

Another related feature in the IIe is the system diagnostics. By pressing the Reset key while simultaneously holding the Control and Solid Apple keys, the built-in self-test

software is operated. This checks the memory and system circuitry. The response "KERNEL OK" indicates all is in order.

Both the II-Plus and the IIe use the Synertek 6502 microprocessor. The IIe uses the 6502A version. The microprocessor has a transfer rate of 8 bits (one character). Available memory is another important issue in computer performance. The II-Plus has a standard 48K bytes of memory, expandable to 64K, and the IIe has 64K, expandable to 128K.

One feature which contributes to the pleasure of working with the Apple Computer is the keyboard (see Figure 1.3). The keys have a very nice "feel" and their arrangement is like a standard typewriter. One flaw in the II-Plus keyboard design, the location of the Reset key, has been remedied in the IIe. A Caps Lock key has been added to the IIe keyboard (unnecessary in the II-Plus since only upper case was available). Tab and Delete keys, along with some new characters, are also included in the IIe keyboard.

The software product that we used in this text is Applesoft BASIC. Integer BASIC is also available with the IIe, but was not used in this text.

1.2 Types of Applications

We will not attempt an exhaustive list of all the applications for which the Apple Computer might be used. It would be impossible and any attempt would require several pages. Instead, we will discuss some general applications and some limitations so that when you encounter a possible application, you will be able to decide whether or not the Apple Computer can be applied.

Although the Apple Computer is a relatively small package, its capabilities can be compared to minicomputers of ten years ago and mainframe computers of fifteen years ago. It is an ideal tool for the engineer or scientist because of its outstanding flexibility. Some of the applications for which it can be used are: (1) a computer for computational work, (2) a terminal to interface to a large mainframe computer, (3) a word processing work station, (4) a process control computer, (5) a controller for a robot, and (6) a data collection device. In addition, the tremendous number of firms developing hardware and software for the Apple Computer will enhance the ways in which it can be applied. Also, competition will insure reasonable prices for the many products developed.

New hardware and software products are continually being announced for the Apple Computer. For example, products have been introduced that permit the Apple Computer to (1) interface with mainframe computers as though it were a terminal or (2) function as a remote job entry terminal (RJE). Before the Apple Computer was available, acquiring

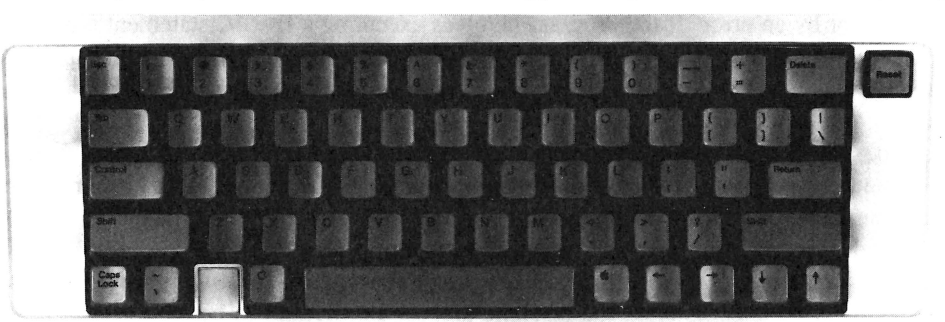


Figure 1.3—Apple IIe Keyboard

a terminal to perform either of these tasks cost considerably more. In addition, these special purpose terminals could not function as a stand-alone computer as can the Apple Computer.

When considering the Apple Computer for a new application, you should be aware of the main limitations of this microcomputer: (1) relatively slow (compared to a minicomputer) when transferring information between memory, the CPU, and other peripheral devices and (2) somewhat slow computational speed. The information transfer rate is slow because only 8 bits (1 character) can be transferred at a time. Another limitation is the small amount of disk storage space; however, this is rapidly changing as disk drives with much greater capacities are being developed. Consequently, the disk storage limitation will essentially be removed.

1.3 The BASIC Interpreter

As noted earlier, the BASIC interpreter used on the Apple Computer is the Applesoft BASIC interpreter. The performance of this interpreter and the differences between interpreters and compilers are discussed in the next section.

Interpreters and Compilers

There are two ways in which a high level programming language, such as BASIC, is translated to *machine language* (instructions that are understood by the computer). One technique uses an *interpreter*; the other technique uses a *compiler*. Many engineers and scientists have worked on a computer with a high level programming language such as FORTRAN. In this environment, the FORTRAN statements are translated into machine language using a compiler. However, Applesoft BASIC statements are translated using an interpreter.

It is important to understand the difference between the two techniques. A compiled program usually executes in less time and is more difficult to alter. The latter advantage provides some protection against unauthorized use of a program. Most microcomputers, however, come with a BASIC interpreter because an interpreter requires less memory than a compiler and is easier to use. These statements are better understood after the differences between an interpreter and compiler are explained.

An interpreter translates one BASIC statement into one or more machine language statements. These machine language statements are then executed. This process continues, one BASIC statement at a time, until execution is terminated by either the end of the program or by an error. If the program involves executing a BASIC statement more than once, the interpreter must translate the BASIC statement into machine language each time the statement is executed because the translated machine language version of the statement is not saved.

A BASIC compiler works somewhat differently. It too translates BASIC statements into machine language. However, a compiler translates the entire BASIC program into machine language statements and stores it in memory before any part of the program is executed. The resulting machine language program is sometimes called the *object* program. The original BASIC version of the program is called the *source* program. After the compilation is completed, the machine language program can be executed.

Once a program has been compiled (translated into machine language), it will run in less time than the same program will take if an interpreter is used. The interpreted program

runs slower because each statement must be interpreted as it is executed; some statements may be translated several times if they are executed several times. However, the interpreter requires less memory because the entire machine language version (object program) does not have to be stored. Program debugging is also easier using an interpreter because execution can be stopped at any point so that a program line can be modified; execution can then be reinitiated at any place in the program.

In writing this book, we assumed that the reader could develop simple programs using BASIC. Many engineers and scientists know the FORTRAN programming language but have never used BASIC. It is very simple to learn BASIC if you already know FORTRAN because these two languages are very similar. There are many good introductory BASIC books written as tutorials to be used as self-study guides. Poole, 1981, and Presley, 1982, are books of this type written just for the Apple Computer.

Review of DOS Commands

When your computer is turned on with a diskette containing the disk operating system (DOS 3.3) in Drive 1, DOS will be loaded into memory. You will know that DOS is active by the following prompt:

```
]
```

This prompt tells you that DOS is waiting for you to enter your next line. The *default drive* is the drive which DOS assumes contains the specified diskette unless another drive is explicitly denoted. The default drive is initially Drive 1 (D1). You can change the default drive to Drive 2 by placing ",D2" after any DOS command. For example

```
]CATALOG,D2
```

followed by pressing the Return key. (Unless otherwise specified, the Return key should be pressed after each command is entered.) This will list the catalog of the diskette in Drive 2, and make it the default drive. Changing the default drive back to Drive 1 is accomplished in a similar manner.

Several commands are provided by DOS. You should look at your *DOS Manual* for a detailed description of these commands.

The DOS CATALOG will list the names of all files contained on a diskette. Included with each name is the file type. For example,

```
]CATALOG
```

will list all files on the diskette in the default drive. The command

```
]CATALOG,D2
```

will list all files on the diskette in Drive 2.

An often used command is LOAD. This command may be used to load a program file. For example,

```
]LOAD EXDATA
```

will load the program file EXDATA from the default drive.

The command

```
]LOAD EXDATA,D2
```

will load the program file from Drive 2. The prompt is displayed after the LOAD command is executed. You can then start program execution by entering

```
]RUN
```

The same results can be achieved by the following statement;

```
]RUN EXDATA
```

This command causes the program EXDATA to load and to begin execution.

After typing in a new program or making changes to one that you originally loaded, you may want to save the program on a diskette. Before doing this, decide on a name to identify the program file. If you select a file name that already exists on the diskette, the file on the diskette by that name will be destroyed. To save the program TESTPROG on the default drive you would type

```
]SAVE TESTPROG
```

When this operation is complete, the prompt will appear. If you purchased the optional diskette with this book, at some time you will probably want to MERGE programs or CHAIN from one program to another. The &H(OLD) and &M(ERGE) commands will let you merge a program stored on a diskette with a program currently in memory. For example, if you purchased the optional diskette, you could merge a subroutine from this diskette with an application program that is currently in memory. There is also a CHAIN statement that permits you to bring another program into memory and continue execution, passing program control and variables, from one program to another if you desire. Using this command, a program that is running can load into memory a program that resides on a diskette and cause this latter program to start executing. We will illustrate the use of the &H and &M commands and the CHAIN statement.

Assume that a sort program resides in a file named SORT on the default drive. This program is to be merged with a program that we currently have in memory. After the prompt appears, the following command should be entered

```
]&H
```

This command holds, or protects, the current program in memory. We can then load the SORT program without wiping-out the program currently in memory.

```
]LOAD SORT
```

In order to merge the original program and the sort program, we use the MERGE command.

```
]&M
```

The process is now complete. When the prompt appears again, the SORT program will be merged with the original program.

Unfortunately, we cannot use the &H and &M commands without some preparation. These commands are enabled by running the program RENUMBER, which is on your APPLE System Master. You must run this program prior to using the commands.

If any lines in the file being merged have the same line numbers as the program in memory, the result will have duplicate line numbers. When the merge has been successfully completed, BASIC displays a prompt.

The CHAIN statement is similar in result to MERGE except that it is entered as part of a program line. Also, after the CHAIN statement is successfuly executed, control is passed to the program that was in the chained (invoked) file rather than BASIC and does not return to the original program.

Two chaining procedures are possible. The first erases the values of all variables from the original program. For example, if you wanted to run the program currently in memory, and then run the program SORT, the last statement in the original program in memory should be

```
PRINT CHR$(4);"RUN SORT"
```

If there were any statements following this one in the original program they would be ignored. No variables from the original program are passed to SORT.

If you wish to pass variables and arrays to the program to which you are chaining, a little additional work is required. A machine-language program called CHAIN is on your DOS System Master. This program must reside on the same diskette as the program to be chained. You can, of course, copy this program from the System Master to any diskette you wish. To chain from the program currently in memory to SORT, the last two lines to be executed in the original program should be:

```
PRINT CHR$(4);"BLOAD CHAIN, A$20"
CALL $20"SORT"
```

Remember, this is not the same as calling a subroutine because you do not return to the calling program. This type of activity would require dividing the main program and placing the sections in separate files. The first part of the main program would chain to the subroutine, which would in turn chain to the next part of the main program.

Using this review, you should be able to load and run the programs that are provided on the optional diskette. There are many other commands and statements which we did not discuss. It was not our intention to provide an introduction to BASIC as there are many good books already available. You can also refer to your *BASIC Programming Reference Manual* if you have questions about Applesoft BASIC.

Variable Types and Precision

Applesoft BASIC variable names may be up to 238 characters long; but only the first 2 characters are significant. They must begin with an alphabetic character. BASIC variables can be of two types: string or numeric. The length of a character string can be from 0 to 255 characters. Two types of numeric variables are permitted: integer and real. Integer variables are distinguished from real variables by placing a percent character, %, after the variable name.

An integer variable can represent any integer between −32768 and +32767, inclusive. Each integer variable requires two bytes of storage. A real variable is used to store a non-integer real number. In this case, 9 digits are stored. Storing a real variable requires 5 bytes.

Loss of Precision

Errors of imprecision occur in most mathematical calculations. However, the imprecision is usually ignored because the accuracy obtained is sufficient. For example, consider the statement

```
X = 3/10
```

The number .3 cannot be represented by any finite length binary number, just as ⅓ cannot be represented by any finite length decimal number. Consequently, mathematical operations on a computer can result in imprecision because of the way numbers are represented.

The following example program will illustrate this point.

```
10  N=40
20  X=3/10
30  PRINT "I", "X"
40  FOR I=2 TO N
50  X=X/10
60  PRINT I,X
70  NEXT I
80  END
```

Figure 1.4 contains the output from this program. Note that for for I=31, X=3.000001E-31. Also, note that for I=40, X=0 and that the computer did not signify an exponent underflow (magnitude of the number is too small).

```
 I  X
 2  .03
 3  3E-03
 4  3E-04
 5  3E-05
 6  3E-06
 7  3E-07
 8  3E-08
 9  3E-09
10  3E-10
11  3E-11
12  3E-12
13  3E-13
14  3E-14
15  3E-15
16  3E-16
17  3E-17
18  3E-18
19  3E-19
20  3E-20
21  3E-21
22  3E-22
23  3E-23
24  3E-24
25  3E-25
26  3E-26
27  3E-27
28  3E-28
29  3E-29
30  3E-30
31  3.00000001E-31
32  3.00000001E-32
33  3.00000001E-33
34  3.00000001E-34
35  3.00000001E-35
36  3.00000001E-36
37  3.00000001E-37
38  3.00000001E-38
```

```
39  3.000000001E-39
40  0
```

Figure 1.4—Loss of Precision Example

The order of mathetical operations can affect precision. Consider the following short program:

```
10  X=5.5E+9
20  Y=X-X+1
30  PRINT Y
```

If this program is executed on a Personal Computer, we will obtain the correct answer of 1. Let us reorder the mathematical operations in statement 20:

```
20  Y=X+1-X
```

Upon executing this modified program, we will obtain an incorrect answer of 2. This program demonstrates that the order of mathematical operation can affect precision, particularly if numbers of widely differing magnitudes are involved.

Converting Variable Types

All arithmetic operations are performed using real precision. Thus, if a program statement involves more than one variable type (integer or real), Applesoft BASIC converts all numbers to real precision. A numeric result is always stored in the precision declared in the target variable name, where the target variable is the variable on the left side of the equal (=) sign. Truncation occurs when a real value is assigned to an integer variable. Thus, I% = 5.5 results in I% = 5.

1.4 Summary

The Apple Computer has several features which make it an excellent microcomputer for engineering, science, and business applications. In addition, it is a pleasure to use. With the support that is being supplied by Apple and independent vendors, numerous types of application hardware and software are available for the Apple Computer at competitive prices. This book is an example of the many programs and texts that are available to Apple users. The remainder of the book presents and explains several programs useful in engineering, science, and business.

References

Poole, Lon, *Apple II User's Guide*, OSBORNE/McGraw-Hill, Berkeley, California, 1981.
Presley, Bruce, *A Guide to Programming in Applesoft*, Van Nostrand Reinhold, New York, 1982.

Exercises

1. What is a source program? What is an object program?
2. What is a BASIC interpreter? What is a BASIC compiler?
3. When would you prefer to use a compiler? When would you prefer to use an interpreter?
4. List five significant features of the Apple Computer.

5. Write a statement that will chain the program EXSMOOTH residing on disk Drive 2 with the program currently in memory.
6. Write a command which will merge the program QUEUE residing on disk Drive 2 (the default drive) with the program currently in memory. Program execution is not to start until the RUN command is entered.
7. Write a program to test the round-off error of the BASIC SIN function as the argument approaches zero (i.e., as X approaches zero, what is the value of SIN(X)?).
8. Obtain the value of X in the following program using all possible variable types for each variable.

```
A=2
B=3
X=A/B
PRINT X
```

9. Let X=50. Write a program that keeps multiplying X by 10 and adding 1 until the 1 is no longer represented in the product (i.e., X=X*10+1). At what power of 10 did you obtain the first incorrect answer?

2

Data Reduction

Data are simply collections of values until they are reduced into a meaningful form. Data are quite often summarized by particular values, called statistics, which are calculated from the data available. These statistics provide valuable information regarding the general location of the data points and their dispersion. This chapter presents a description of the most commonly used statistics and BASIC programs for their calculation. These will prove useful in all data gathering exercises, such as experimentation and observation.

2.1 Basics of Data Reduction

It is difficult to discern meaning from a large collection of data without summarization. The procedures presented in this chapter provide such summarization techniques. We will examine both reduction procedures (statistics) and display procedures. They are not computationally difficult, but provide extremely useful results for data analysis.

One of the methods most commonly used to represent a group of data is to calculate statistics. In this way, particular characteristics of the data can be reduced to one number. Thus, a statistic contains information from the data in a sample. By definition, any function of the data is a statistic. However, it is important to choose a function that results in a statistic which is meaningful from an analysis viewpoint. We will limit our discussion to only a few common statistics. Population statistics reflect on the entire universe of data under consideration. For instance, if we were examining automobiles, the population would be all automobiles manufactured. Sample statistics attempt to measure the same thing as population statistics, but they are found from only a subset of the population. They should give a reasonably accurate representation or estimate of the population statistic.

There are four major reasons why we are concerned with sample statistics. Often we must use the data we are given and are not in a position to gather more. Data collection could be expensive. Collecting performance data on rocket engines is an expensive proposition. Data collection could be destructive. When collecting data on the life of a light bulb, the light bulb is destroyed. Finally, if there are an infinite number of points in a population, it is impossible to examine them all.

In this chapter we examine two general categories of statistical measures; these are measures of central tendency and measures of dispersion. Central tendency relates to the

center of the data. The statistic calculated in this case represents, in some fashion, the center of the data. Figures 2.1 and 2.2 help to illustrate this concept. If you were to select one number to reflect the "center" of data for each of Experiments A and B, you would choose two different numbers. The center of Experiment A, in Figure 2.1, appears to be around 25 while the center of Experiment B, in Figure 2.2, appears to be around 30. You will see shortly how this can be measured in several ways.

Dispersion relates to the spread of the data. That is, are the data points spread across a wide range of values or are they bunched together? Figures 2.1 and 2.2 also illustrate dispersion. The data in Figure 2.1 are more dispersed than those in Figure 2.2. We will illustrate several ways in which this can be measured.

Although there may be many ways in which we can summarize data through the use of statistics, we must not underestimate the value of simply looking at the data. This examination goes beyond looking at a group of numbers on a page. It should include a visual inspection of a plot of the data. In this manner it is much easier to detect grouping of data items and the spread of data items. In many instances, examination of the data in this fashion can prove as beneficial as calculation of the statistics.

The remainder of this chapter is divided into three major sections. Each section addresses an analysis area described above. First, we will present measures of central tendency, then measures of dispersion, and finally operational aspects of plotting data.

2.2 Measures of Central Tendency

The most common statistics used are those representing central tendency. These statistics are used to reflect the center of the data. Those discussed here each measure the "center" of the data in a slightly different way. The choice of a measure depends on the particular situation in which it is being applied.

Mean

The arithmetic mean of a group of data is also known as the average. We can think of the mean as the center of gravity of the data. If each point weighed the same amount and they were placed along a weightless number line, the mean would be that point at which the number line would be balanced. It will not, in general, be equal to any particular data point.

Finding the mean of a group of data is straightforward; you have probably done it before. All one has to do is sum the data points and divide by the number of data points. We can represent this by:

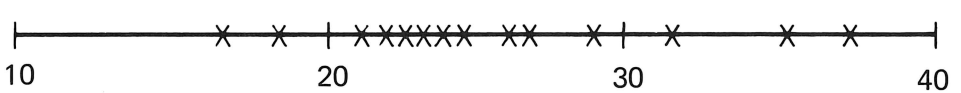


Figure 2.1—Plot of Data From Experiment A

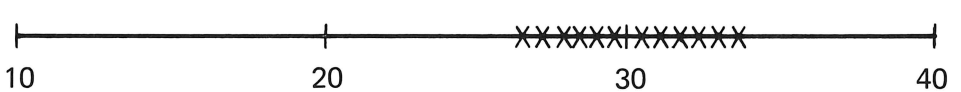


Figure 2.2—Plot of Data From Experiment B

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

where,

X_i = value for data point i

n = number of data points

$\bar{X}$ = mean or average (sometimes called X-bar)

For a small amount of data, this can easily be accomplished by hand or performed on a calculator.

EXAMPLE 1. Find the mean of the following group of numbers:

35, 21, 42, 16, 71, 55

SOLUTION.

$$\bar{X} = \frac{35+21+42+16+71+55}{6} = \frac{240}{6} = 40$$

For larger groups of data this becomes tedious; obviously it is a job for the computer. The only tasks required of a computer program are to read in the data and make the calculation to find the mean. We do not present a subroutine here to find the mean. Such a subroutine is presented, with the other measures discussed, in a complete data analysis subroutine later in this chapter.

Median

The median is another measure of central tendency. It differs from the mean in that it is not the "center of gravity" of the data. Instead, it is a point on the line which "splits" the data. That is, the same number of data points are on either side of the median.

The median of a group of data is easily found. If we rank the data points in ascending order, we can count from either end until we have counted half the data points. At this point we have determined the median. There are two conditions we may face. If there is an odd number of data points, then one of the data points will be the median. There will be one data point that has the same number of data points on either side of it. In this case the median is unique. The other condition is an even number of data points. In this case none of the data points may be the median. If there are n data points then there will be n/2 on either side of the median. Thus, if the data are ranked, the median will lie between the n/2 and (n/2) + 1 data points. Theoretically, any point between these can be classified as the median. For our purposes, we will choose the point equidistant between the n/2 and (n/2) + 1 points. This is the average, or mean, of these two points.

EXAMPLE 2. Find the median of the following set of data.

3, 9, 2, 15, 22, 11, 7, 35, 4, 16

Compare this to the value of the mean.

SOLUTION. First, rank the data in ascending order.

Rank	1	2	3	4	5	6	7	8	9	10
Data	2,	3,	4,	7,	9,	11,	15,	16,	22,	35

There are $n = 10$ data points, so the median will be the average of the $n/2 (= 5)$ and $(n/2) + 1 (= 6)$ points. Thus,

$$\text{median} = X_m = \frac{9+11}{2} = 10.$$

We can calculate the mean as

$$\overline{X} = \frac{2+3+4+7+9+11+15+16+22+35}{10} = 12.4$$

Of course there is a difference between the median and the mean, although they are close. You can probably see why the median is sometimes a more useful measure than the mean. The mean is affected by the size of the data and the possibility that large numbers do not reflect the rest of the data. In this case, most of the data is less than 20, but the point 35 has a large impact on the mean, whereas its impact on the median is not nearly as large. Thus, either may be more appropriate in particular situations.

Mode

The least-used measure of central tendency is the mode. The mode is the data point which occurs most often. In the first two examples of this chapter, the mode is each data point, since no points are repeated. There may be only one mode, or many modes as in the examples seen thus far.

One would not want to use the mode as the sole measure of central tendency. It should be used with the mean or the median, or both.

EXAMPLE 3. Find the mode of the following set of data.

4, 2, 8, 10, 14, 8, 6, 14, 20, 6, 3, 5, 8, 6

Also find the mean and the median.

SOLUTION. To find the mode, like the median, it is helpful to rank the data in ascending order. Doing this we find:

2, 3, 4, 5, 6, 6, 6, 8, 8, 8, 10, 14, 14, 20

The values 6 and 8 each appear three times, more than any other values. These would both represent the mode of the data.

For the median, notice that there are 14 data points; $n = 14$. The median is the average of the seventh and eighth ranked points. So:

$$X_m = \frac{6+8}{2} = 7$$

We calculate the mean as:

$$\bar{X} = \frac{\sum_{i=1}^{14} X_i}{14} = \frac{114}{14} = 8.1429$$

It will not always be the case that the mean, median, and mode are in such close agreement as in the above examples. This is why it is often beneficial to examine all three measures. For instance, look at the difference in measures for the following set of data.

Data:	3, 4, 4, 5, 2, 4, 6, 5, 3, 20, 22, 20, 24, 26, 20, 25
Mean:	$\bar{X} = 12$
Median:	$X_m = 5$
Mode:	4 and 20

You can see that each by itself does not convey the structure of the data. By examining all measures you get a pretty good feel for the content of the data.

2.3 Measures Of Dispersion

The measures of central tendency just discussed describe the way in which data are clustered around some central value. They do not tell us how close or how far apart the data values are; that is, how they are scattered. A statistic measuring the scatter or spread of a group of data is called a measure of dispersion. Two different groups of data may have identical means, medians, and modes, yet be vastly different because of differences in dispersion.

In this section we will discuss three measures of dispersion: range, variance, and standard deviation. The latter two measures are closely related and we will present them together.

Range

The range of a group of data is a simple measure both to understand and calculate. It is defined as the difference between the largest and smallest values in the data. You can see that this gives us a number which specifies the "width" of the data points. It does not contain any additional information that may be desirable, such as where the data points are concentrated.

EXAMPLE 4. Find the range for the following group of data:

3, 6, 2, 9, 5, 3, 16, 4, 15

SOLUTION. If we rank the data in ascending order, finding the range becomes a simple matter of finding the difference between the end points.

2, 3, 3, 4, 5, 6, 9, 15, 16
Range $= X_R = 16 - 2 = 14$

Although the range is quite easy to calculate, in most cases it does not provide sufficient information about the variation of the data. Because only two data points are used, the range says nothing about the location and concentration of other data points except the

distance between them is less than or equal to X_R. You have also noticed that the range does not depend on the number of data points. For these reasons, we now present two more measures of dispersion. These will provide more useful information than the range.

Variance and Standard Deviation

The variance and the standard deviation are quite different in concept and calculation from the range. They are based upon the distance of each individual point from the mean. Because of this, their calculation is significantly more involved than that for the range.

In using these measures for more advanced statistical procedures, either the variance or the standard deviation will be specified for use. While they certainly are not the same measure, they are very closely related. We find the value of the standard deviation as the positive square root of the variance. We will, therefore, focus our discussion on the variance.

We mentioned above that the variance was based upon the distance of each individual point from the mean. For data point i this deviation is $X_i - \overline{X}$. You may notice that if we were to sum the deviations for all of the data points, the sum would be zero. We, therefore, cannot use the measure "average deviation from the mean;" its value would be zero. Since we would like to consider any deviation from the mean as positive, we could square the deviations, thus assuring that they would be positive. We could also use the absolute value of deviations from the mean, instead of the square of deviations. However, this option is not as mathematically attractive and does not possess some desirable characteristics. Consult DeGroot (1975) for further discussion.

The variance is defined as the average or mean of the squares of the deviations from the mean. By utilizing the definition of the mean, we can define the variance symbolically as

$$S^2 = \frac{\sum_{i=1}^{n} (X_i - \overline{X})^2}{n}$$

You may notice a slightly different definition of variance in other sources. Using the above definition the variance is called a "biased" estimate of the true population variance. An "unbiased" estimate is found by dividing by $n-1$ instead of n. You can consult Walpole and Myers (1972) or DeGroot (1975) for a discussion of estimator bias.

EXAMPLE 5. What is the variance of the following set of data?

3, 6, 8, 2, 4, 3

SOLUTION. Following the formula for the variance, we first must calculate $\overline{X}$.

$$\overline{X} = \frac{\sum_{i=1}^{6} X_i}{6} = \frac{3+6+8+2+4+3}{6} = 4.33$$

The variance can now be found as:

Introducing —

Diskette to Accompany BASIC Engineering, Science and Business Programs for the Apple II and IIe

Phillip M. Wolfe and C. Patrick Koelling

Now you can select and access any number of programs — *quickly and easily* — because all the keyboarding has been done for you!

Just look at what you'll get:

- Fast access to linear programming and regression, next-event simulation, project planning and scheduling, forecasting with exponential smoothing, and more.
- Hours and hours of your valuable time saved. (After all, isn't that the main reason why you bought your Apple?)
- No more exasperating glitches and keystroke errors. And that means a clear head and keen eye when you're ready to get down to some serious programming.

Here's How To Order

Enclose a check or money order for $25.00, plus sales tax, slip in this handy order envelope and mail! No postage needed. Or charge it to your VISA or MasterCard. Simply complete the information below.

ORDER TODAY!

☐ **YES!** I want to quickly and easily access 38 major programs! Please rush me **Diskette to Accompany BASIC Engineering, Science and Business Programs for the Apple II and IIe**/D2883-9. I have enclosed payment of $25.00 plus sales tax.

Name _____

Address _____

City _____ State_____ Zip_____

Dept. Y

Charge my Credit Card Instead

☐ VISA ☐ MasterCard

_____
Account Number

_____
Expiration Date

_____
Signature as it appears on Card

BRADY

Brady Communications Co., Inc.
A Prentice-Hall Publishing Company
Bowie, Maryland 20715

of the units of the data. If the data are in units of miles, the variance will be in units of square miles. For this reason the standard deviation may be a more desirable measure of the dispersion. Since the standard deviation is the square root of the variance, its units are the same as those of the original data. We label the standard deviation S:

$$S = \sqrt{S^2}$$

In a straightforward manner the standard deviation of the data in Example 5 is found to be $S = \sqrt{4.222} = 2.055$. The computer code for these measures of dispersion is presented with the other data analysis code later in this chapter.

Give Yourself The Gift Of Time

And get virtually *instant* access to 38 major engineering, science and business programs . . .

See over for details!

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1976 BOWIE, MD

POSTAGE WILL BE PAID BY ADDRESSEE

Brady Communications Co., Inc.
A Prentice-Hall Publishing Company
Bowie, Maryland 20715

3, 6, 8, 2, 4, 3

SOLUTION. Following the formula for the variance, we first must calculate $\overline{X}$.

$$\overline{X} = \frac{\sum_{i=1}^{6} X_i}{6} = \frac{3+6+8+2+4+3}{6} = 4.33$$

The variance can now be found as:

$$S^2 = \frac{(3-4.33)^2+(6-4.33)^2+(8-4.33)^2+(2-4.33)^2+(4-4.33)^2+(3-4.33)^2}{6}$$

$$S^2 = 4.222$$

Computationally, this formula for S^2 is not very attractive. By rearranging terms we can find a much easier one to apply. This formula is

$$S^2 = \frac{\sum_{i=1}^{n} X_i^2 - \frac{\left(\sum_{i=1}^{n} X_i\right)^2}{n}}{n}$$

As with the previous definition, this is the "biased" estimate of the population variance.

EXAMPLE 6. Use the revised formula to find the variance of the data in Example 5.

SOLUTION. All we need do is find the sum of the values and the sum of the square of the values.

X_i	X_i^2
3	9
6	36
8	64
2	4
4	16
3	9
$\sum_{i=1}^{6} X_i = 26$	$\sum_{i=1}^{6} X_i^2 = 138$

The variance is:

$$S^2 = \frac{138 - \frac{(26)^2}{6}}{6} = 4.222$$

You can see that if the data is centered around the mean, the variance is very small. If there are many data points which are not close to the mean, the variance will be large. The larger the variance, the more scattered the data.

You should note the units of measurement of the variance. The units will be the square of the units of the data. If the data are in units of miles, the variance will be in units of square miles. For this reason the standard deviation may be a more desirable measure of the dispersion. Since the standard deviation is the square root of the variance, its units are the same as those of the original data. We label the standard deviation S:

$$S = \sqrt{S^2}$$

In a straightforward manner the standard deviation of the data in Example 5 is found to be $S = \sqrt{4.222} = 2.055$. The computer code for these measures of dispersion is presented with the other data analysis code later in this chapter.

2.4 Data Plots

The measures discussed previously are valuable in reducing a particular property of data to one number. However, it is quite often just as valuable to examine a visual representation of the data. This can be accomplished by plotting the data on a graph. This plot can show the values of the data and the frequency of occurrence. If the data consist of joint observations, they can also be represented. For instance, you may have monthly sales data over a number of years. It would be useful to examine a plot of sales over time.

One of the most important advantages of looking at the data as opposed to simply calculating statistics is examining joint data. Here joint data refers to the case when a single observation results in values for two variables. An example of this is the observation of a person's physical stature. Observing one person results in the joint measure of height and weight. We do not get an accurate picture of physical stature unless we look at the height/weight combination. Another example is measuring metal fatigue. In this case, it is important to look at the joint observations of force and time to failure.

Variables may be related in many different ways. It is important to be able to specify these relationships. In some cases, trend or seasonal patterns can be detected. There are analytic procedures for evaluating these types of patterns, but they are not presented here (see Makridakis and Wheelwright, 1978, and Montgomery and Johnson, 1976, for analytic procedures). You have not sufficiently examined a set of data unless you have examined a plot of the data.

In the computer program presented in the next section, we include a plotting routine. You will be able to specify single or joint data. An example of a data plot is not included in this section.

2.5 Data Analysis and Plotting Subroutines

In this section, we present a set of computer subroutines that incorporate all the data analysis procedures discussed in the chapter. There are actually three major subroutines. The first reduces the data to particular statistics. These are the measures of central tendency and measures of dispersion presented earlier. This program provides the capability to call the plotting subroutines. In these subroutines a curve may be plotted or a frequency histogram (specifying frequency of occurrence of observations) of a set of data can be generated. These plots were written to generate plots for the 40-column display or the 80-column display available with the Apple IIe. The 80-column plots were tested on the 80-column board provided with the IIe. Due to differences in hardware design, the 80-column option may or may not work with other 80-column boards.

Figure 2.3 presents the subroutine for data analysis. This allows analysis of either a single variable or joint observations. The measures of central tendency and dispersion are displayed for each variable.

Figure 2.4 presents the frequency plotting subroutine, including a companion subroutine for plotting on your printer. It can be called from the data analysis program or from a main program, by-passing the data analysis. It will plot a frequency histogram for one variable. You can choose 40 or 80 column formats.

Figure 2.5 presents the subroutine for plotting joint observations. It can also be used with or without the data analysis subroutine. The option of 40 or 80 column is available.

```
7500  REM      SUBROUTINE FOR DATA ANALYSIS
7510  REM      STATISTICS ARE DERIVED IN THIS MAIN PROGRAM,
7520  REM      A SEPARATE SUBROUTINE PLOTS THE DATA.
7540 N = INUM
7541 DOS$ =  CHR$ (4)
7550  PRINT : PRINT "IS THIS ANALYSIS FOR ONE VARIABLE OR"
7551  PRINT "JOINT OBSERVATIONS ?"
7560  PRINT : INPUT "ENTER 1-- ONE VARIABLE; 2-- JOINT ? ";IA
7570  IF IA = 1 THEN  GOTO 7750: REM     FOR THE ONE -VARIABLE CASE
7580  REM          ********** RULES FOR JOINT OBSERVATIONS **********
7590  REM
7600  PRINT : PRINT "THE DATA FOR THIS PROGRAM MUST BE"
7601  PRINT : PRINT "ENTERED AS JOINT OBSERVATIONS IN THE"
7610  PRINT : PRINT "VARIABLES X AND Y WITH THE NUMBER OF"
7620  PRINT : PRINT "JOINT OBSERVATIONS GIVEN IN INUM. THE"
7630  PRINT : PRINT "OBSERVATIONS ARE SORTED IN A SPECIALLY"
7631  PRINT : PRINT "REVISED QUICKSORT ROUTINE THAT"
7640  PRINT : PRINT "MAINTAINS THE INTEGRITY OF THE JOINT"
7650  PRINT : PRINT "RELATIONSHIPS FOR VALUES OF JT<>3 ."
7660 JT = 1: REM     SET VARIABLE SPECIFICATION FLAG
7670  PRINT : PRINT "SORTING DATA": GOSUB 12710: REM     CALL THE REVISED SORT
7680  FOR I = 1 TO INUM
7690 XVAR(I) = X(I): REM     SET XVAR
7700 YVAR(I) = Y(I): REM     SET YVAR
7710  NEXT I
7720  GOTO 7800
7730  REM          ********** RULES FOR FREQUENCY ANALYSIS **********
7740  REM
7750  PRINT : PRINT "THE DATA FOR THIS PROGRAM MUST BE"
7751  PRINT : PRINT "PRE-SORTED IN ASCENDING ORDER. THE"
7760  PRINT : PRINT "SORTED VALUES MUST APPEAR IN THE ARRAY"
7770  PRINT : PRINT "XVAR. THE NUMBER OF DATA POINTS MUST"
7780  PRINT : PRINT "APPEAR IN INUM"
7790 JT = 0: REM     SET FOR FREQUENCY DATA
7800  REM     IN THIS SECTION THE DATA IS ANALYZED
7810  REM     ACCORDING TO THE STATISTICS PRESENTED
7820  REM     IN THE CHAPTER (MODE,MEDIAN,MEAN,
7830  REM     RANGE,STANDARD DEVIATION,VARIANCE)
7840  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Z$
7850  PRINT : PRINT
7860  IF JT <  > 2 THEN  PRINT "FOR VARIABLE ";XV$
7870  IF JT = 2 THEN  PRINT "FOR VARIABLE ";YV$
7871  PRINT
7880  PRINT  TAB( 6);"MEASURES OF CENTRAL TENDENCY"
7881  PRINT  TAB( 5);"=============================="
7900  REM     FIRST FIND THE MODE
7910 J = 1: REM    SET MODE COUNTER
7920 JNUM = 0: REM     NUMBER OF DATA POINTS REPRESENTING THE MODE
7930 JCNT = 2: REM     DATA POINT OCCURRENCE COUNTER
7940 JJ = 0
7950  IF JT = 2 THEN  GOTO 7970
7960  DIM ILOC(INUM): REM     DIMENSION FOR LOCATION OF MODE
7970  FOR I = 2 TO INUM + 1
7980  IF I = INUM + 1 THEN  GOTO 8020
7990  IF XVAR(I) <  > XVAR(I - 1) THEN  GOTO 8020: REM    FOR UNEQUAL VALUES
8000 J = J + 1: REM     UPDATE FREQUENCY COUNTER
8010  GOTO 8180
8020  IF J >  = JCNT THEN  GOTO 8050: REM    FREQUENCY LESS THAN CURRENT MAX
8030 J = 1
8040  GOTO 8180
8050  IF J = JCNT THEN  GOTO 8140: REM    FREQUENCY EQUAL TO CURRENT MAX
8060  REM    AT THIS POINT,THE CURRENT FREQUENCY REPRESENTS THE MAXIMUM
8070  REM    FREQUENCY. UPDATE ALL PERTINENT COUNTERS
8080 JNUM = 1
8090 JCNT = J
8100 J = 1
```

```
8110 JJ = 1
8120 ILOC(JNUM) = I - 1: REM    SET TO LOCATE MODE
8130  GOTO 8180
8140 JNUM = JNUM + 1: REM    SET FOR MULTIPLE MODES
8150 ILOC(JNUM) = I - 1
8160 J = 1
8170 JJ = 1
8180  NEXT I
8190  IF JJ = 1 THEN  GOTO 8220: REM    A MODE DOES EXIST
8200  PRINT : PRINT "MODE -- NO OBSERVATION APPEARED MORE"
8201  PRINT "THAN ONCE"
8210  GOTO 8320
8220  IF JNUM > 1 THEN  GOTO 8250: REM    MORE THAN ONE MODE EXISTS
8230  PRINT : PRINT "MODE -- THE VALUE ";XVAR(ILOC(1));" OCCURRED ";JCNT;" TIMES"
8240  GOTO 8320
8250  PRINT : PRINT "MODE -- MULTIPLE MODES WERE FOUND"
8260  PRINT "         THEY ALL OCCURRED ";JCNT;" TIMES"
8270  PRINT "         THESE VALUES ARE :"
8280  FOR I = 1 TO JNUM
8290  IF I = JNUM THEN  PRINT XVAR(ILOC(I))
8300  IF I <  > JNUM THEN  PRINT XVAR(ILOC(I));", ";: REM    PRINT THE MULTIPLE MODES
8310  NEXT I
8320  REM    NEXT, FIND THE MEDIAN
8340 XX = INUM / 2
8350 II =  INT (INUM / 2)
8360  IF II = XX THEN  GOTO 8390: REM    INTEGER NUMBER OF DATA POINTS
8370  PRINT : PRINT "MEDIAN -- ";XVAR(II + 1)
8380  GOTO 8410
8390 MED = (XVAR(II) + XVAR(II + 1)) / 2
8400  PRINT : PRINT "MEDIAN -- ";MED
8410  REM      NOW, CALCULATE THE MEAN
8430 SM = 0: REM    SET SUM OF TERMS EQUAL TO ZERO
8440 SS = 0: REM    SET SUM OF SQUARED TERMS EQUAL TO ZERO
8450  FOR I = 1 TO INUM
8460 SM = SM + XVAR(I): REM    SUM THE TERMS
8470 SS = SS + XVAR(I) ^ 2: REM    SQUARE AND SUM THE TERMS
8480  NEXT I
8490 MN = SM / INUM: REM   FIND THE MEAN
8500  PRINT : PRINT "MEAN -- ";MN: PRINT
8520  PRINT  TAB( 9);"MEASURES OF DISPERSION"
8521  PRINT  TAB( 8);"=========================="
8540  REM    FIRST FIND THE RANGE
8550 RNG = XVAR(INUM) - XVAR(1): REM    CALCULATE THE RANGE
8560  PRINT : PRINT "RANGE -- ";RNG
8570  REM    NOW FIND THE VARIANCE AND STANDARD DEVIATION
8590 VAR = (SS - ((SM ^ 2) / INUM)) / INUM: REM   CALCULATE THE VARIANCE
8600 STD =  SQR (VAR): REM   FIND THE STANDARD DEVIATION
8610  PRINT : PRINT "VARIANCE -- ";VAR
8620  PRINT
8630  PRINT "STANDARD DEVIATION -- ";STD
8640  PRINT
8650  IF IA = 1 THEN  GOTO 8890
8660  IF JT = 2 THEN  GOTO 8830
8670 XRNG = RNG
8680 JT = 2: REM    RESET VARIABLE SPECIFICATION FLAG
8690 XT = XVAR(1): REM   SET XT EQUAL TO SMALLEST XVAR VALUE
8700 XM = XVAR(INUM): REM   SET XM EQUAL TO LARGEST XVAR VALUE
8710  FOR I = 1 TO INUM
8720 X(I) = YVAR(I): REM   EXCHANGE XVAR AND YVAR ARRAYS,PREPARE FOR SORT
8730 Y(I) = XVAR(I)
8740  NEXT I
8750  REM    NOW CALL REVISED SORTING ROUTINE TO SORT YVAR AND
8760  REM    KEEP THE RELATIONSHIP BETWEEN XVAR AND YVAR
8770  PRINT : PRINT "SORTING DATA": GOSUB 12710: REM   CALL THE REVISED SORT ROUTINE
8780  FOR I = 1 TO INUM
8790 XVAR(I) = X(I): REM    REASSIGN XVAR
```

```
8800 YVAR(I) = Y(I): REM    REASSIGN YVAR
8810  NEXT I
8820  GOTO 7800
8830  FOR I = 1 TO INUM
8840 C = XVAR(I)
8850 XVAR(I) = YVAR(I): REM   EXCHANGE XVAR AND YVAR ARRAYS
8860 YVAR(I) = C
8870  NEXT I
8880 YRNG = RNG
8890  PRINT : INPUT "DO YOU WANT A PLOT OF THE DATA (Y/N) ? ";B$
8910  IF B$ = "N" THEN  GOTO 8960
8920  IF IA = 1 THEN  GOTO 8950
8930  GOSUB 10960
8940  GOTO 8960
8950  GOSUB 8970: REM    GO TO THE PLOTTING SUBROUTINE
8960  RETURN
8970  REM   ************************************************************
```

Figure 2.3—Data Analysis Program

Both the data analysis and the plot routines require that the data be sorted from smallest to largest. You may enter them in this fashion if you wish. However, we find it is easier to enter the data conveniently and use a sorting subroutine to order them. Figure 2.6 presents a sorting subroutine which is included in the program file. We will not discuss how it works, because this is done in Chapter 12. However, it does have one very important attribute. If the flag JT = 3, then normal sorting of one variable from smallest to largest occurs. If the flag JT≠3, then the sort of variable X from smallest to largest also preserves the appropriate relationships of the joint observations. That is, each Y variable remains associated with its particular X variable. This is essential for plotting joint observations.

An important aspect of the plotting subroutine for a frequency histogram is the choice of an integer or non-integer plot. If the non-integer option is selected, class intervals are automatically established. In this case, instead of plotting the frequency of each data point, the frequency of occurrence of data points within specified intervals is plotted. This will be the option you use most. If the integer option is selected (your data must be integer values), two possibilities exist. If the range of the data is less than 60 (24 for the 40-column case), the frequency for each point will be plotted. If the range is greater than 60 (24 for the 40 column-case), class intervals are used.

The data analysis and plotting subroutines were not written as main programs, so they could be more flexible in application. Therefore, a main program must be written in order to use them. There are a few important steps which must be taken in the calling program. They are specified in the program listing but are worthy of presentation here. First, general requirements applicable to all subroutines will be presented, then requirements specific to each subroutine.

General

1. Data must be stored in variable XVAR (and YVAR for joint observations).
2. Number of observations must be stored in INUM.
3. Name of variable(s) must be stored in XV$ for XVAR (and YV$ for YVAR if required).
4. Appropriate arrays must be dimensioned to INUM.

```
8980  REM     FREQUENCY PLOT SUBROUTINE
8990  REM     THIS IS THE FREQUENCY PLOTTING SUBROUTINE.
9000  REM     IT CAN BE USED TO PLOT A FREQUENCY HISTOGRAM FOR A SINGLE
9010  REM     SET OF DATA.
9020  REM     INPUT ARRAY IS XVAR AND MUST BE DIMENSIONED
9030  REM     IN THE MAIN PROGRAM. ALSO REQUIRED IS THE NUMBER OF DATA
9040  REM     POINTS, SORTED IN INUM.
9050 ID = 1
9060 XM = XVAR(INUM)
9070 RNG = XVAR(INUM) - XVAR(1): REM   SET RANGE
9080  PRINT : PRINT "IS THIS FREQUENCY PLOT FOR INTEGER OR"
9081  PRINT "NONINTEGER DATA ?"
9090  PRINT : INPUT "ENTER D -- INTEGER, C -- NONINTEGER ? ";A$
9100  PRINT : INPUT "PLOT ON CRT OR PRINTER (C/P) ? ";C$: PRINT
9110  IF C$ = "P" THEN  PRINT "PLEASE TURN ON PRINTER": PRINT
9111  INPUT "FORTY OR EIGHTY COLUMN PLOT (F/E)?";S$
9112  IF S$ = "E" THEN  GOTO 9115
9113 C% = 24:R% = 6:W% = 4:S% = 11:F% = 50
9114 T% = 10:WD% = 30: GOTO 9120
9115 C% = 60:R% = 10:W% = 6:S% = 12:F% = 75
9116 T% = 20:WD% = 70
9120  IF C$ = "P" THEN  INPUT " HIT CARRIAGE RETURN WHEN READY ? ";WQ$
9121  IF C$ = "P" THEN  PRINT DOS$;"PR#1"
9130  IF A$ = "D" THEN  GOTO 10390
9140  REM    THIS SECTION IS FOR NONINTEGER DATA
9150  GOSUB 12330: REM    SCALE THE VARIABLES
9160 XW = RNG / C%
9170 IC = R%: REM   USE 6 CLASSES TO STORE THE DATA
9180  REM    CALCULATE FREQUENCY OF OCCURRENCE IN EACH CLASS
9190  DIM FQ(IC + 1): REM    DIMENSION FREQUENCY ARRAY
9200  FOR I = 1 TO INUM
9210  IF I = 1 THEN J = 1
9220  IF XVAR(I) > XVAR(1) + J * (RNG / IC) + .0001 THEN  GOTO 9250
9230 FQ(J) = FQ(J) + 1: REM    INCREASE THE FREQUENCY BY ONE
9240  GOTO 9270
9250 J = J + 1: REM    GO TO THE NEXT CLASS
9260  GOTO 9220: REM    CHECK THE NEXT CLASS
9270  NEXT I
9280 IMF = 1: REM    SET MAXIMUM FREQUENCY
9290  FOR I = 1 TO IC
9300  IF FQ(I) > IMF THEN IMF = FQ(I): REM    SET MAXIMUM FREQUENCY
9310  NEXT I
9320  PRINT : PRINT : PRINT : PRINT
9340  POKE 36,( - 1 + T%): PRINT "FREQUENCY PLOT OF ";XV$: PRINT
9370  REM    DETERMINE FREQUENCY SCALE
9380  DIM IVA(15): REM    DIMENSION LABEL ARRAY
9390  IF IMF > 15 THEN  GOTO 9490
9400 IS =  INT (15 / IMF): REM   SET NUMBER OF LINES BETWEEN FREQUENCIES
9410 LN = IS * IMF: REM    SET NUMBER OF LINES IN PLOT
9420 J = 1: REM    SET START FOR VERTICAL COUNTER
9430 II = IS
9440  FOR I = II TO LN STEP IS
9450 IVA(I) = J: REM    SET VERTICAL AXIS LABEL
9460 J = J + 1
9470  NEXT I
9480  GOTO 9540
9490 IS =  INT ((IMF / 15) + 1): REM    SET NUMBER OF FREQUENCIES PER LINE
9500 LN =  INT ((IMF / IS) + 1): REM    SET NUMBER OF LINES IN PLOT
9510  FOR I = 1 TO LN
9520 IVA(I) = I * IS: REM    SET VERTICAL AXIS LABEL
9530  NEXT I
9540 IQ = LN: IF LN < 9 THEN LN = 9
9550  FOR I = 1 TO LN
9580  IF I > 9 THEN  GOTO 9680
9590  IF I = 1 THEN  PRINT " F";
```

```
9600  IF I = 2 THEN  PRINT " R";
9610  IF I = 3 THEN  PRINT " E";
9620  IF I = 4 THEN  PRINT " Q";
9630  IF I = 5 THEN  PRINT " U";
9640  IF I = 6 THEN  PRINT " E";
9650  IF I = 7 THEN  PRINT " N";
9660  IF I = 8 THEN  PRINT " C";
9670  IF I = 9 THEN  PRINT " Y";
9680 IJ = LN - I + 1: IF IJ > IQ THEN  GOTO 9820
9690  IF IVA(IJ) <  > 0 THEN IK = IVA(IJ)
9700  IF IVA(IJ) = 0 THEN  POKE 36,( - 1 + 9): PRINT "+";
9710  IF IVA(IJ) <  > 0 THEN  POKE 36,( - 1 + 4): PRINT IVA(IJ);
9730  IF IVA(IJ) <  > 0 THEN  POKE 36,( - 1 + 9): PRINT "+";
9740  FOR J = 1 TO IC
9750  IF ID <  > 2 AND J = 1 THEN  POKE 36,( - 1 + 9 + J * W%): PRINT "I";
9760  IF FQ(J) < IK THEN  GOTO 9790
9770  IF ID = 2 THEN  POKE 36,( - 1 + 9 + J * IXW): PRINT "*";
9780  IF ID <  > 2 THEN  POKE 36,( - 1 + S% + J * W%): PRINT "*";
9790  IF ID = 2 THEN  GOTO 9810
9800  IF J <  > IC THEN  POKE 36,( - 1 + 9 + (J + 1) * W%): PRINT "I";
9810  NEXT J
9820  PRINT
9830  NEXT I
9840  REM   PRINT HORIZONTAL AXIS
9850  POKE 36,( - 1 + 9): PRINT "+";
9870  IF ID = 2 THEN  GOTO 9930
9880  FOR I = 0 TO IC
9890  IF S$ = "E" THEN  PRINT "-----+";
9900  IF S$ = "F" THEN  PRINT "---+";
9910  NEXT I
9920  GOTO 10030
9930  FOR I = 1 TO IXW * (RNG + 1) + 1
9940 IG = 0: REM    SET PRINTER FLAG
9950 IH =  INT (I / IXW)
9960  IF IH = (I / IXW) THEN IG = 1
9970  IF IG = 1 THEN  PRINT "+";
9980  IF IG = 0 THEN  PRINT "-";
10010  NEXT I
10020 IC = R%: REM   RESET INC FOR INTEGER CASE
10030  PRINT
10050  REM    PRINT AXIS VALUES
10060  IF ID = 2 THEN  GOTO 10180
10070  IF XI% = 0 THEN  GOTO 10081
10080  POKE 36,( - 1 + 1): PRINT "(10^";XI%;")";
10081  IF S$ = "E" THEN  GOTO 10126
10082  FOR I = 0 TO IC STEP 2
10084 HAX = XVAR(1) + I * W% * XW
10086 ZB = HAX / XE
10088 ZB =  INT (ZB * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
10090  GOSUB 12850: REM   SET FORMAT
10092  IF K% = 0 THEN J% = 14 -  LEN ( STR$ (ZB))
10094  IF K% <  > 0 THEN J% = 13 - K%
10096  POKE 36,( - 1 + J% + I * W%)
10098  PRINT ZB;
10100  NEXT I
10102  PRINT
10104  FOR I = 1 TO IC STEP 2
10106 HAX = XVAR(1) + I * 4 * XW
10108 ZB = HAX / XE
10110 ZB =  INT (ZB * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
10112  GOSUB 12850: REM   SET FORMAT
10114  IF K% = 0 THEN J% = 14 -  LEN ( STR$ (ZB))
10116  IF K% <  > 0 THEN J% = 13 - K%
10118  POKE 36,( - 1 + J% + I * 4)
```

```
10120  PRINT ZB;
10122  NEXT I
10124  GOTO 10310
10126  FOR I = 0 TO IC
10128 HAX = XVAR(1) + I * 6 * XW
10130 ZB = HAX / XE
10132 ZB =  INT (ZB * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
10134  GOSUB 12850: REM   SET FORMAT
10136  IF K% = 0 THEN J% = 16 -  LEN ( STR$ (ZB))
10138  IF K% <  > 0 THEN J% = 15 - K%
10140  POKE 36,( - 1 + J% + I * 6)
10142  PRINT ZB;
10144  NEXT I
10146  GOTO 10310
10180 JI = XVAR(1)
10181  IF S$ = "E" THEN WD% = 70
10182  IF S$ = "F" THEN WD% = 30
10190  FOR I = 1 TO WD%
10200  IF I = 1 THEN  GOTO 10241
10210  IF I < J THEN  GOTO 10300
10220  IF  INT ((I - 1) / IXW) <  > (I - 1) / IXW THEN  GOTO 10300
10230 JI = XVAR(1) + (I - 1) / IXW
10240  IF JI > XVAR(INUM) THEN  GOTO 10300
10241 R$ =  STR$ (JI)
10261  POKE 36,( - 1 + 9 + IXW + I -  LEN (R$))
10270  PRINT JI;
10290 J = I + 5
10300  NEXT I
10310  REM    PRINT HORIZONTAL AXIS TITLE
10370  PRINT :
10371  IF S$ = "F" THEN  POKE 36,( - 1 + 12)
10372  IF S$ = "E" THEN  POKE 36,( - 1 + 35)
10373  PRINT XV$
10380  GOTO 10590
10390  REM    FOR INTEGER DATA
10400  REM   IF THE RANGE IS > C%, CLASSES MUST BE USED
10410  IF RNG > C% - 1 THEN  GOTO 9150
10420 ID = 2: REM     SET INTEGER FLAG
10430 IC = C%: REM   SET NUMBER OF DATA POINTS
10440 IXW =  INT (C% / (RNG + 1)): REM   COLUMNS PER DATA POINT
10450  DIM FQ(F%)
10460 J = 1: REM    SET COORDINATE COUNTER
10461  REM   SCALE LARGE INTEGER VALUES
10462 GG =  INT (XVAR(1) / 1000) * 1000
10463  FOR I = 1 TO INUM
10464 XVAR(I) = XVAR(I) - GG
10465  NEXT I
10470  FOR I = 1 TO INUM
10480  IF XVAR(I) <  > XVAR(1) + (J - 1) THEN  GOTO 10510
10490 FQ(J) = FQ(J) + 1: REM    FIND FREQUENCY FOR EACH COORDINATE
10500  GOTO 10550
10510 J = J + 1
10520  IF J > C% + 1 THEN  GOTO 10570
10530  IF XVAR(I) <  > XVAR(1) + (J - 1) THEN  GOTO 10510
10540 FQ(J) = FQ(J) + 1: REM    FIND FREQUENCY FOR EACH COORDINATE
10550  NEXT I
10551 DQ = J
10552 IC = DQ
10560  GOTO 9280: REM    GO TO THE MAIN PLOTTING STAGE
10570  PRINT : PRINT "PROGRAM DATA ERROR; PROGRAM TERMINATING"
10580  GOTO 10670
10590  IF C$ = "P" THEN  GOTO 10670
10600  PRINT : PRINT "WOULD YOU LIKE THE PLOT ON YOUR PRINTER"
10601  INPUT "(Y/N) ? ";D$
10610  IF D$ = "N" THEN  GOTO 10670
10620  PRINT : PRINT "PLEASE TURN ON THE PRINTER"
```

```
10630  PRINT : INPUT "HIT CARRIAGE RETURN WHEN READY ? ";WQ$
10640 C$ = "P"
10641  PRINT DOS$;"PR#1"
10650  IF ID = 2 THEN IC = DQ
10660  GOTO 9550
10670  PRINT DOS$;"PR#0": RETURN
```

Figure 2.4—Frequency Plot Subroutine

Single Observations: Data analysis

1. XVAR must be sorted from smallest to largest.

Joint Observations: Data analysis

1. Variables X and Y must also be dimensioned to INUM.
2. Data need not be sorted in the main program.

Frequency Plot

1. Data must be sorted from smallest to largest.

Joint Observation Plot

1. Set XT as the smallest number in the XVAR array.
2. Set XM as the largest number in the XVAR array.
3. YVAR must be sorted from smallest to largest.
4. XVAR variables must match with appropriate joint observations in YVAR (i.e., their position in the array must be the same).

We should mention the format used for the plots. In order to create an accurate yet readable plot, a specified number of spaces was allowed to print a number and the position of the decimal point. Because of this, the size of the numbers which can be plotted is restricted. Restrictions on XVAR, for both plotting subroutines, allow numbers from 0.01 to 99.99 (9999 for integer frequency plots). For YVAR (joint observations) the restriction is 0.01 to 999.99. If your numbers are larger or smaller than those allowed, they will automatically be scaled to fit this format (except integer frequency plots). A subroutine is included for this purpose, and is presented in Figure 2.7.

Since there is a limited number of rows and columns for plotting, a single row or column may actually represent many values, as though intervals were used. You must keep this in mind when interpreting the plots. When plotting joint observations, different plot characters are used. If one spot on the plot represents only one observation, the character "*" is printed. If more than one observation is represented, the number of observations represented is printed.

Finally, you may elect to create your plot on the CRT or on your printer. When creating an 80-column plot on the printer, the computer must be set in the 80-column mode.

Let us look at examples of the data analysis and plotting subroutines. We will first examine frequency plots and then analysis of joint observations. The following example illustrates the use of the data analysis and plotting routines.

EXAMPLE 7. Write a main program to use the data analysis program to analyze and plot the following data.

3.6, 4.2, 1.1, 12.5, 8.4, 6.3, 10.1, 0.8, 5.6, 6.5
7.4, 3.8, 9.2, 7.5, 8.8, 6.3, 1.0, 9.6, 10.2, 13.1

```
10970  REM    SUBROUTINE FOR PLOTTING JOINT OBSERVATIONS
10980  REM    SUBROUTINE FOR PLOTTING TWO VARIABLRS
10990  REM    DATA CONSISTS OF JOINT OBSERVATIONS,XVAR AND YVAR;
11000  REM    THE RANGES OF EACH VARIABLE, XRNG AND YRNG. THE
11010  REM    DATA YVAR MUST BE SORTED IN ASCENDING ORDER. THE
11020  REM    NUMBER OF JOINT OBSERVATIONS IS INUM.
11030  REM    THE NAMES OF THE VARIABLES ARE ASSUMED TO BE
11040  REM    GIVEN IN THE VARIABLES XV$ AND YV$.
11050  REM    THE VALUE XT MUST BE SPECIFIED AS THE SMALLEST VALUE
11060  REM    OF THE XVAR ARRAY. THIS IS ASSIGEND IN THE DATA ANALYSIS
11070  REM    SUBROUTINE.
11080  REM    IN SUMMARY, THIS IS WHAT IS REQUIRED :
11090  REM          1. YVAR SORTED IN ASCENDING ORDER
11100  REM          2. XVAR JOINT OBSERVATION (APPROPRIATE SUBSCRIPTS
11110  REM             MUST MATCH)
11120  REM          3.  THE NUMBER OF JOINT OBSERVATIONS IN INUM
11130  REM          4.  NAMES FOR XVAR AND YVAR (XV$ AND YV$)
11140  REM          5.  XT - SMALLEST VALUE IN THE XVAR ARRAY
11150  REM          6.  XM -  LARGEST VALUE IN THE XVAR ARRAY
11160  REM          7.  XVAR,YVAR,X,AND Y, MUST BE DIMENSIONED TO
11170  REM              INUM IN THE CALLING PROGRAM
11171  INPUT "FORTY OR EIGHTY COLUMN PLOT (F/E)?";S$
11172  IF S$ = "E" THEN  GOTO 11176
11173 C% = 24:H% = 5:T% = 15:A% = 4
11175  GOTO 11180
11176 C% = 60:H% = 11:T% = 30:A% = 10
11180  PRINT : PRINT "WOULD YOU LIKE THE PLOT ON YOUR PRINTER"
11181  INPUT "(Y/N) ? ";D$
11190  IF D$ = "Y" THEN  PRINT : PRINT "PLEASE TURN ON YOUR PRINTER"
11200  IF D$ = "Y" THEN  PRINT : INPUT "PRESS CARRIAGE RETURN TO BEGIN ? ";Q$
11201  IF D$ = "Y" THEN  PRINT DOS$;"PR#1"
11210 JT = 3: REM    SET FLAG FOR NORMAL SORTING
11220  REM    THESE MUST BE PROVIDED IF THE DATA ANALYSIS ROUTINE IS NOT
11230  REM    USED TO ACCES THIS PLOTTING SUBROUTINE.
11240 XSTP = XRNG / C%: REM   FIND HORIZONTAL STEP RANGE
11250 YSTP = YRNG / 15: REM    FIND VERTICAL STEP RANGE
11260  DIM SOX(INUM)
11270  REM    REARRANGE XVAR AND YVAR SO YVAR IS IN DESCENDING ORDER
11280  FOR I = 1 TO INUM
11290 SOX(I) = XVAR(I)
11300  NEXT I
11310  FOR I = 1 TO INUM
11320 YVAR(I) = X(INUM - I + 1): REM   X HAS YVAR VALUES FROM PREVIOUS SORT
11330 XVAR(I) = SOX(INUM - I + 1)
11340  NEXT I
11350  GOSUB 12330: REM     SCALE THE YVAR AND XVAR VARIABLES
11360 JL =  LEN (YV$): REM    NUMBER OF CHARACTERS IN YV$
11370  FOR I = 0 TO 15
11380 AL = 0: REM    INITIALIZE VAL
11390 VX = YVAR(1) - I * YSTP
11400 LOWER = VX - YSTP / 2: REM    SET LOWER CELL LIMIT
11410  PRINT
11430  IF I > JL - 1 THEN  GOTO 11470
11440  IF I = 0 THEN F$ =  LEFT$ (YV$,1)
11450  IF I <  > 0 THEN F$ =  MID$ (YV$,I + 1,1)
11470  IF I < JL THEN  PRINT F$;
11480  POKE 36,( - 1 + 3)
11481 ZB = VX / YE + 0.000001
11482 ZB =  INT (ZB * 10 ^ 2 + 0.5) /  INT (10 ^ 2 + 0.5)
11490  PRINT ZB;
11500  POKE 36,( - 1 + 10): PRINT "+";
11560  IF I = 0 THEN IT = 1: REM    SET YVAR COUNTER
11570  FOR J = IT TO INUM
11580  IF YVAR(J) < LOWER THEN  GOTO 11610
11590 AL = AL + 1
```

```
11600  NEXT J
11610  IF AL = 0 THEN  GOTO 11940
11620  REM    AT LEAST ONE VALUE FOR ROW I
11630  IF AL = 1 THEN  GOTO 11720: REM    SORTING IS NOT NECESSARY
11640  FOR J = IT TO IT + AL - 1
11650 X(J - IT + 1) = XVAR(J): REM    ASSIGN X ARRAY FOR SORT
11660  NEXT J
11670 N = AL
11680  GOSUB 12710: REM    CALL SORT ROUTINE
11690  FOR J = IT TO IT + AL - 1
11700 XVAR(J) = X(J - IT + 1): REM    REASSIGN XVAR
11710  NEXT J
11720 JY = IT
11730 JVA = 1
11740  FOR J = JY TO IT + AL - 1
11750  IF J = JY THEN IOL =  INT (((XVAR(JY) - XT) / XSTP) + 0.000001)
11760  IF J = JY THEN  GOTO 11800
11770 JOL =  INT (((XVAR(J) - XT) / XSTP) + 0.000001): REM   FIND APPROPRIATE COLUMN
11780  IF JOL <  > IOL THEN  GOTO 11810: REM    NOT IN SAME COLUMN AS PREVIOUS
11790 JVA = JVA + 1: REM    INCREASE OBSERVATIONS IN COLUMN
11800  NEXT J
11810 PCH$ = "*"
11820  IF JVA = 1 THEN  GOTO 11870
11830 PCH$ =  STR$ (JVA)
11840  POKE 36,( - 1 + 14 + IOL): PRINT PCH$;
11860  GOTO 11890
11870  POKE 36,( - 1 + 15 + IOL): PRINT PCH$;
11890 JY = JY + JVA: REM    UPDATE COUNTER FOR THIS ROW
11900  IF JY - IT + 1 > AL THEN  GOTO 11920: REM    NO MORE OBSERVATIONS IN THIS ROW
11910  GOTO 11730
11920 IT = IT + AL: REM    RESET YVAR VARIABLE COUNTER
11930  IF IT > INUM THEN  GOTO 11960
11940  NEXT I
11960  PRINT
11970  IF YI% <  > 0 THEN  POKE 36,( - 1 + 1): PRINT "(10^";YI%;")";
11980  POKE 36,( - 1 + 10): PRINT "+"
11990  POKE 36,( - 1 + 10): PRINT "+";
12000  PRINT "-----+";
12070  FOR I = 1 TO A%
12080  PRINT "-----+";
12100  NEXT I
12120  PRINT : PRINT
12130  IF XI% <  > 0 THEN  POKE 36,( - 1 + 3): PRINT "(10^";XI%;")";
12170  FOR I = 1 TO H%
12180 HAX = XT + (I - 1) * 6 * XSTP: REM    FIND HORIZONTAL AXIS VALUE
12190 ZB = HAX / XE
12191 ZB =  INT (ZB * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
12192 K% = 0
12193  FOR IC = 1 TO 4
12194 E$ =  MID$ ( STR$ (ZB),IC,1)
12195  IF E$ = "." THEN K% = IC - 1
12196  NEXT IC
12197  IF K% <  > 0 THEN  GOTO 12201
12198 K% =  LEN ( STR$ (ZB)) - 1
12199  POKE 36,( - 1 + 16 + (I - 1) * 6 - K%)
12200  GOTO 12202
12201  POKE 36,( - 1 + 15 + (I - 1) * 6 - K%)
12202  PRINT ZB;
12250  NEXT I
12270  PRINT : PRINT
12280  POKE 36,( - 1 + T%): PRINT XV$
12320  PRINT DOS$;"PR#0": RETURN
12330  REM   ************************************************************
```

Figure 2.5—Joint Observations Plot Subroutine

```
12711  REM    SUBROUTINE: QUICKSORT (REVISED)
12712  REM    THIS SUBROUTINE USES THE QUICKSORT PROCEDURE TO SORT A LIST
12713  REM    OF NUMBERS INTO ASCENDING ORDER.THE LIST MUST BE PROVIDED TO
12714  REM    THIS SUBROUTINE IN THE ARRAY X. THE NUMBER OF ELEMENTS IN THE
12715  REM    LIST MUST BE SUPPLIED IN THE VARIABLE N.THE ARRAY LEF AND RIT
12716  REM    ARE USED TO STORE SUBSETS TO BE PARTITIONED. LEF AND RIT
12717  REM    CONTAIN THE POSITION OF THE FIRST ELEMENT AND THE LAST ELEMENT
12718  REM    OF A SUBSET TO BE PARTITIONED.LS AND RS MAINTAIN THE LEFT AND
12719  REM    RIGHT END POSITIONS OF THE SUBSET BEING PARTITIONED. L AND R
12720  REM    INDICATE THE ELEMENTS THAT HAVE BEEN COMPARED WITH THE PIVOT,
12721  REM    PV. PT INDICATES THE POSITION IN THE ARRAY LEF AND RIT OF THE
12722  REM    LAST SUBSET STORED TO BE PARTITIONED.IF ALL THE REM STATEMENTS
12723  REM    ARE REMOVED FROM WITHIN THE SORT PROCEDURE,THE SUBROUTINE WILL
12724  REM    SORT A LIST IN LESS TIME.
12725  REM    CHANGING THE > IN LINE 12764 TO < AND CHANGING THE < IN
12726  REM    LINE 12768 TO > WILL RESULT IN THE ARRAY BEING SORTED IN
12727  REM    DECREASING ORDER. SET TR%=0 FOR FIRST PASS THROUGH.
12728  REM    SIZ% = 50
12729  IF TR% = 1 THEN  GOTO 12731: REM    BYPASS DIMENSION
12730  DIM LEF(SIZ%),RIT(SIZ%)
12731 LEF%(1) = 1: REM    LEFT END OF SUBSET
12732 RIT%(1) = N: REM    RIGHT END OF SUBSET
12733 TR% = 1: REM    SET TO RECORD PASS
12734  REM   POSITION IN LEF AND RIT OF LAST SUBSET STORED
12735 PT% = 1
12736  REM    SET LEFT SIDE POINTER
12737 L% = LEF%(PT%)
12738  REM    SAVE STARTING VALUE OF LEFT SIDE POINTER
12739 LS% = L%
12740  REM    SET RIGHT SIDE POINTER
12741 R% = RIT%(PT%)
12742  REM    SAVE STARTING VALUE OF RIGHT SIDE POINTER
12743 RS% = R%
12744 PT% = PT% - 1
12745  REM    IF ONLY 1 ELEMENT IN SUBSET, DO NOT PARTITION
12746  IF (R% - L%) <  = 0 THEN 12817
12747  REM    TEMPORARILY SET PIVOT TO LEFT ELEMENT
12748 PV = X(L%): IF JT <  > 3 THEN PY = Y(L%)
12749  REM    IF 5 OR LESS ELEMENTS IN SUBSET LEAVE PIVOT AS IS
12750  IF (R% - L%) < 5 THEN 12763
12751  REM    SET PIVOT TO MEDIAN
12752 MED = (L% + R%) / 2
12753  IF (PV < X(R%)) AND (PV > X(MED)) THEN 12763
12754  IF (PV < X(MED)) AND (PV > X(R%)) THEN 12763
12755  IF (X(R%) < PV) AND (X(R%) > X(MED)) THEN 12761
12756  IF (X(R%) < X(MED)) AND (X(R%) > PV) THEN 12761
12757  REM    SET PIVOT = MEDIAN AND MOVE LEFT POINTER TO HOLE
12758 PV = X(MED): IF JT <  > 3 THEN PY = Y(MED)
12759 X(MED) = X(L%): IF JT <  > 3 THEN Y(MED) = Y(L%)
12760  GOTO 12763
12761 PV = X(R%): IF JT <  > 3 THEN PY = Y(R%)
12762 X(R%) = X(L%): IF JT <  > 3 THEN Y(R%) = Y(L%)
12763  IF L% >  = R% THEN 12779: REM    IFL = R, SUBSET PARTITIONED
12764  IF PV > X(R%) THEN 12771: REM   IF TRUE SWAP
12765 R% = R% - 1: REM   MOVE POINTER
12766  GOTO 12763
12767  IF L% >  = R% THEN 12779: REM   IF L=R, SUBSET PARTITIONED
12768  IF PV < X(L%) THEN 12775: REM   IF TRUE,SWAP
12769 L% = L% + 1: REM    MOVE POINTER
12770  GOTO 12767
12771 X(L%) = X(R%): REM    PERFORM SWAP
12772  IF JT <  > 3 THEN Y(L%) = Y(R%)
12773 L% = L% + 1: REM    MOVE POINTER
12774  GOTO 12767
12775 X(R%) = X(L%): REM   PERFORM SWAP
```

```
12776  IF JT < > 3 THEN Y(R%) = Y(L%)
12777 R% = R% - 1: REM     MOVE POINTER
12778  GOTO 12763
12779 X(L%) = PV: REM     PLACE PIVOT IN HOLE
12780  IF JT < > 3 THEN Y(L%) = PY
12781  REM     DETERMINE WITCH SUBSET TO STORE;LARGEST IS TO BE STORED
12782  IF (L% - LS%) < = (RS% - L%) THEN 12800
12783  REM     RIGHT SUBSET IS SMALLEST;IF IT CONTAINS MORE THAN 1
12784  REM     ELEMENT, STORE LEFT SUBSET
12785  IF (RS% - L%) < = 1 THEN 12795
12786 LEF%(PT% + 1) = LS%
12787 RIT%(PT% + 1) = L% - 1
12788 L% = L% + 1
12789 LS% = L%
12790 R% = RS%
12791 PT% = PT% + 1
12792  GOTO 12746
12793  REM     RIGHT SUBSET CONTAINED ONLY 1 ELEMENT;PARTITION LEFT
12794  REM     SUBSET IF IT CONTAINS MORE THAN 1 ELEMENT
12795  IF (L% - LS%) < = 1 THEN 12817
12796 R% = L%I1
12797 RS% = L%
12798 L% = LS%
12799  GOTO 12746
12800  IF (L% - LS%) < = 1 THEN 12810
12801  REM     LEFT SUBSET IS SMALLEST;IF IT CONTAINS MORE THAN 1
12802  REM     ELEMENT, STORE RIGHT SUBSET
12803 LEF%(PT% + 1) = L% + 1
12804 RIT%(PT% + 1) = RS%
12805 R% = L% - 1
12806 RS% = R%
12807 L% = LS%
12808 PT% = PT% + 1
12809  GOTO 12746
12810  IF (RS% - L%) < = 1 THEN 12817
12811  REM     RIGHT SUBSET CONTAINED MORE THAN 1 ELEMENT;PARTITION IT
12812 L% = L% + 1
12813 LS% = L%
12814 R% = RS%
12815  GOTO 12746
12816  REM     ANY MORE SUBSETS TO BE PARTITIONED ?
12817  IF (PT% < 1) THEN 12819
12818  GOTO 12737
12819  RETURN
```

Figure 2.6—Sorting Subroutine

```
12340  REM     SCALING SUBROUTINE :
12355  REM     THIS SUBROUTINE IS USED FOR SCALING THE NUMBERS USED IN THE
12360  REM     PLOTS TO FIT THE FORMAT SPECIFICATIONS. THIS SUBROUTINE
12370  REM     SHOULD NOT BE USED WITHOUT ACCESSING IT THROUGH THE PLOTTING
12380  REM     SUBROUTINE
12390  IF IA = 1 THEN  GOTO 12550: REM   SCALE ONLY THE XVAR VARIABLE
12400  REM     FOR YVAR SCALING
12410  IF YVAR(1) < 1 THEN  GOTO 12470: REM     SCALE VARIABLE UP
12420  FOR I = 0 TO 35
12430 YI% = I
12440 YE = 10 ^ I
12450  IF YVAR(1) / YE < 100 THEN  GOTO 12550
12460  NEXT I
12470  REM     FOR YVAR LESS THAN ONE
12480  FOR I = 3 TO 35
```

```
12490 YI% =  - I
12500 YE = 10 ^ YI%
12510  IF YVAR(1) / YE > 100 THEN  GOTO 12530
12520  NEXT I
12530 YE = YE * 10: REM    SET APPROPRIATE VALUE OF YE
12540 YI% = YI% + 1
12550  REM    FOR SCALING THE XVAR VALUES
12560  IF XM < 1 THEN  GOTO 12620
12570  FOR I = 0 TO 35
12580 XI% = I
12590 XE = 10 ^ I
12600  IF XM / XE < 100 THEN  GOTO 12700
12610  NEXT I
12620  REM    FOR XVAR  LESS THAN ONE
12630  FOR I = 3 TO 35
12640 XI% =  - I
12650 XE = 10 ^ XI%
12660  IF XM / XE > 100 THEN  GOTO 12680
12670  NEXT I
12680 XE = XE * 10
12690 XI% = XI% + 1
12700  RETURN
12710  REM   ************************************************************
```

Figure 2.7—Scaling Subroutine

SOLUTION. The main program used to access the subroutines is listed in Figure 2.8. The statistical summary for this group of data is given below. The frequency histogram is presented in Figure 2.9 for a 40-column plot.

```
                ***MEASURES OF CENTRAL TENDENCY***
MODE -- THE VALUE 6.3 OCCURRED 2 TIMES
MEDIAN -- 6.95
MEAN -- 6.8

                   ***MEASURES OF DISPERSION***
RANGE -- 12.3
VARIANCE -- 12.29
STANDARD DEVIATION -- 3.50571
```

```
310  REM MAIN PROGRAM TO READ IN SINGLE OBSERVATIONS
320  REM THIS PROGRAM CALLS DATA ANALYSIS (AND PLOTTING SUBROUTINES
330  REM IF DESIRED). A REVISED SORT ROUTINE IS USED FOR
340  REM SORTING THE DATA.
360  INPUT "ENTER THE NUMBER OF OBSERVATIONS TO BE ANALYZED";INUM
370  DIM XVAR(INUM),X(INUM)
380  INPUT "ENTER THE NAME OF THE VARIABLE";XV$
390  PRINT "ENTER THE OBSERVATIONS ONE AT A TIME"
400  FOR I=1 TO INUM
410  PRINT
420  PRINT "ENTER OBSERVATION";I;
430  INPUT X(I)
440  NEXT I
450  JT=3: REM  SET FLAG FOR BUBBLE SORT ROUTINE
451  N=INUM
460  GOSUB 12710: REM  CALL SORTING ROUTINE
470  FOR I=1 TO INUM
480  XVAR(I)=X(I):  REM  PLACE SORTED VALUES IN XVAR
490  NEXT I
500  GOSUB 7500
510  END
```

Figure 2.8—Main Program For Frequency Plots

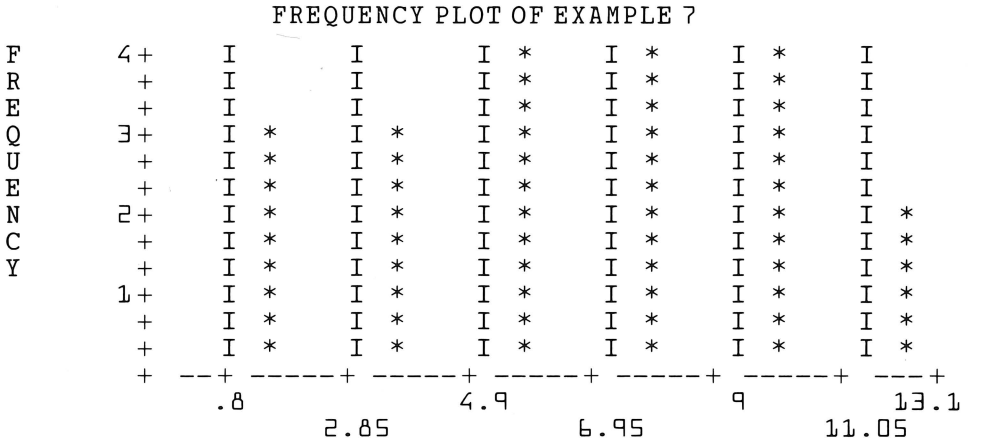


Figure 2.9—Example 7, Frequency Histogram

EXAMPLE 8. Write a main program to analyze and plot the following joint observations.

X:	3	6	9	4	5	3	1	2	8	6	7	10
Y:	4	5	10	4	7	5	1	4	10	5	9	14

SOLUTION. The main program required is presented in Figure 2.10. The data analysis output is presented below and the joint observation plot is given in Figure 2.11 for a 40-column plot.

```
FOR VARIABLE X
***MEASURES OF CENTRAL TENDENCY***

MODE -- MULTIPLE MODES WERE FOUND
        THEY ALL OCCURRED 2 TIMES
        THESE VALUES ARE
 3 , 6
MEDIAN -- 5.5
MEAN -- 5.333334

***MEASURES OF DISPERSION***

RANGE -- 9
VARIANCE -- 7.3888889
STANDARD DEVIATION -- 2.718251

FOR VARIABLE Y
***MEASURES OF CENTRAL TENDENCY***

MODE -- MULTIPLE MODES WERE FOUND
        THEY ALL OCCURRED 3 TIMES
        THESE VALUES ARE
 4 , 5
MEDIAN -- 5
MEAN -- 6.5
```

```
***MEASURES OF DISPERSION***


RANGE -- 13
VARIANCE -- 11.91667
STANDARD DEVIATION -- 3.452053


100  REM MAIN PROGRAM TO READ IN JOINT OBSERVATIONS
110  REM PROGRAM CALLS DATA ANALYSIS AND PLOTTING SUBROUTINES.
120  REM A REVISED SORT ROUTINE IS USED FOR MAINTAINING
130  REM PROPER RELATIONSHIPS BETWEEN JOINT OBSERVATIONS.
140  REM THIS IS CALLED FROM THE DATA ANALYSIS SUBROUTINE.
160  INPUT "ENTER THE NUMBER OF JOINT OBSERVATIONS";INUM
170  DIM XVAR(INUM),YVAR(INUM),X(INUM),Y(INUM)
180  INPUT "ENTER THE NAME OF THE FIRST VARIABLE OF THE PAIR";XV$
190  INPUT "ENTER THE NAME OF THE SECOND VARIABLE OF THE PAIR";YV$
200  PRINT "ENTER THE JOINT OBSERVATIONS"
210  FOR I=1 TO INUM
220  PRINT
230  PRINT "OBSERVATION ";I
240  PRINT "FOR ";XV$;
250  INPUT X(I)
260  PRINT "FOR ";YV$;
270  INPUT Y(I)
280  NEXT I
290  GOSUB 7500
300  END
```

Figure 2.10—Main Program for Plotting Joint Observations

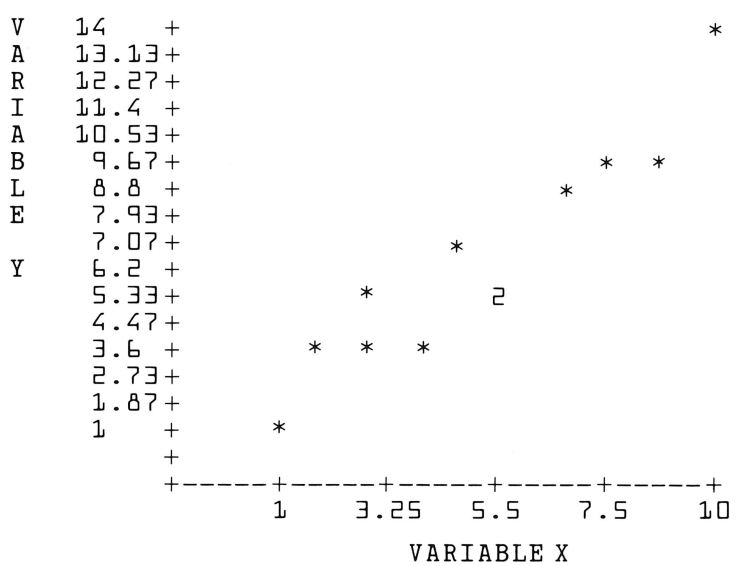


Figure 2.11—Joint Observation Plot

2.6 Summary

This chapter has concentrated on the analysis and display of data. We are always being bombarded with data, but they are useless unless they can be reduced to a meaningful level through statistics or data plots. We have covered the most commonly used statistics and presented a plotting tool through which to examine the data visually. We discussed measures of central tendency (mean, median, and mode) and measures of dispersion (range, variance, and standard deviation). This information, and the plot, can be used with other chapters in this book, in particular Chapters 15 (Random Numbers) and 4 (Curve Fitting).

References

DeGroot, Morris H. *Probability and Statistics*, Addison-Wesley Publishing Company, Inc., Reading, Mass., 1975.

Freund, John E. *Mathematical Statistics*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1962.

Gray, A. William and Otis M. Ulm *Elementary Probability and Statistics*, Glencoe Press, Beverly Hills, Calif., 1973.

Makridakis, Spyros and Steven C. Wheelwright *Forecasting: Methods and Applications*, John Wiley and Sons, New York, N.Y., 1978.

Montgomery, Douglas C. and Lynwood A. Johnson *Forecasting and Time Series Analysis*, McGraw-Hill Book Company, New York, N.Y. 1976.

Walpole, Ronald E. and Raymond H. Myers *Probability and Statistics for Engineers and Scientists*, The Macmillan Company, New York, N.Y., 1972.

Exercises

1. Following is a collection of computer response times (in seconds) from a remote facility. Process these through the data reduction and plotting subroutines presented in this chapter.

3.2	1.0	8.5
6.1	3.7	3.0
1.3	0.6	2.8
1.2	2.1	4.6
4.6	0.9	3.3
2.3	5.3	1.1
5.1	4.4	3.2

2. Two metals are being considered for use in a new product. The most important property under consideration is tensile strength. You must make the decision as to which metal to use. Use the following test data to help support your decision. The data consist of tensile strength measurements (in pounds per square inch) from several tests.

Metal A

9500	9632
9300	9513
9650	9481
9525	9490
9420	9691
9803	9510

Metal B

9210	9506
10320	9005
8925	8985

9610	10145
8760	9200
10010	9600

3. Following is a set of joint data regarding percent sunlight received by a plant during the day and growth per day in centimeters. Use the plotting routine to plot this joint data.

Percent Sunlight	*Growth*	*Percent Sunlight*	*Growth*
25	3	85	6
40	5	50	4
30	3	30	3
50	5	75	6
95	8	10	3
60	5	30	4
80	6	15	2
40	4	70	6

   Does it appear that sunlight is the only factor relevant to plant growth?
4. Give cogent arguments for using each of the mean, median, and mode as measures of central tendency. Give specific examples of when one may be preferred over the others.
5. From problem 3, we may wish to estimate plant growth based upon the percent of sunlight received. Draw a straight line through your plot from problem 3, that line which you feel "best" represents the data. (There are formal methods for this procedure discussed in Chapter 4.) What growth would you expect for 55 percent sunlight? What is the equation of your line?

3

Matrices and Vectors

Many mathematical formulas of interest in engineering, science and business, employ matrices. Matrices improve computational efficiency and provide a convenient way to represent complex relationships. This chapter details the use of matrices and vectors on the Apple Computer. Many basic mathematical operations are described, including single-matrix manipulations.

3.1 Definition of a Matrix

A matrix itself does not have any mathematical meaning. It is simply a table of numbers arranged in a specific manner. The numbers are arranged in columns (or rows) of equal length. The general structure is

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ \vdots & & & & \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}$$

The matrix A above has m rows and n columns, and is therefore classified as an $m \times n$ (read "m by n") matrix. We always list the number of rows first. If $m = n$, we say that the matrix is "square." Thus, the matrix B is a 3×2 matrix.

$$B = \begin{bmatrix} 1 & 2 \\ 3 & 6 \\ 1 & 8 \end{bmatrix}$$

Within a computer, a matrix is usually stored in an array. The number of rows and columns for each array must be specified. We do this through the DIMENSION (DIM) statement. An example of a DIM statement is

```
DIM A(2,4), B(5,2)
```

This statement specifies variables A and B as arrays. For each array variable we specify the number of rows first and the number of columns second. Thus, A is a 2×4 array and B is a 5×2 array. This dimensioning procedure ensures that the computer will allocate space for and recognize the use of the specified arrays. The computer does not require that you use all of the space allocated in the DIM statement. Matrix A may be only a 2×2. The DIM statement places only upper limits on array size.

Individual elements of array variables are referenced in a manner similar to the way the arrays are dimensioned in the DIM statement. That is, an array element is identified by its row and column position within the array. The element in the second row and third column of the array A is denoted A(2,3). The following is a specification of an array in computer terms.

$$A = \begin{bmatrix} 1 & 2 & 3 & 3 \\ 2 & 1 & 8 & 6 \end{bmatrix}$$

A(1,1)=1	A(1,2)=2	A(1,3)=3	A(1,4)=3
A(2,1)=2	A(2,2)=1	A(2,3)=8	A(2,4)=6

This notation is important to understand because it is used throughout the computer programs presented in this chapter. It is also necessary for reading a matrix. A subroutine for this exercise is presented in Figure 3.1.

```
13000  REM ************************************************************
13010  REM     SUBROUTINE TO READ IN THE DIMENSIONS OF A MATRIX,
13020  REM    THE MATRIX ITSELF, AND ALLOCATE SPACE THROUGH DIM
13030  INPUT "THE ROW DIMENSION OF A1 IS ? ";IROW
13040  INPUT "THE COLUMN DIMENSION OF A1 IS ? ";ICOL
13050  DIM A1(IROW,ICOL): REM          DIMENSION THE MATRIX A1
13060  PRINT : PRINT "INPUT THE MATRIX A1 1 ELEMENT AT A TIME,
       ROW-BY-ROW": PRINT
13070  FOR I = 1 TO IROW
13080  FOR J = 1 TO ICOL
13090  INPUT A1(I,J)
13100  NEXT J,I
13110  RETURN
13120  ***(repeat)
13130  SUBROUTINE TO READ IN THE DIMENSIONS OF TWO MATRICES,
13140  A1 AND B1, AND ALLOCATE SPACE THROUGH DIM
13150  INPUT "THE ROW DIMENSION OF A1 IS";IROW
13160  INPUT "THE COLUMN DIMENSION OF A1 IS";ICOL
13170  INPUT "THE ROW DIMENSION OF B1 IS";JROW
13180  INPUT "THE COLUMN DIMENSION OF B1 IS";JCOL
13190  DIM A1 (IROW, ICOL), B1 (JROW, JCOL)
13200  PRINT "ENTER THE MATRIX A1 ONE ELEMENT AT A TIME, ROW-BY-ROW"
13210  FOR I-1 TO IROW
13220  FOR J-1 TO ICOL
13230  INPUT A1 (I,J)
13240  NEXT J, I
13250  PRINT "ENTER THE MATRIX B1 ONE ELEMENT AT A TIME, ROW-BY-ROW"
13260  FOR I-1 TO JROW
13270  FOR J-1 TO JCOL
13280  INPUT B1 (I, J)
```

```
13290  NEXT J, I
13300  RETURN
```

Figure 3.1—Matrix Dimensions Subroutine

3.2 Matrix Operations

There are a number of basic matrix operations that you will find useful. These operations are detailed below.

Transpose

The transpose of a matrix is a simple concept. It is accomplished by exchanging the rows and columns of a matrix. Row 1 becomes column 1, row 2 becomes column 2, etc. An m × n matrix becomes an n × m matrix when transposed. The transpose is signified by the superscript T.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

The matrices A and A^T have the same elements, but they are arranged in a slightly different order. Lets look at an example:

$$B = \begin{bmatrix} 3 & 6 & 1 \\ 2 & 5 & 0 \\ 2 & 1 & 0 \end{bmatrix} \quad B^T = \begin{bmatrix} 3 & 2 & 2 \\ 6 & 5 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

We present a BASIC subroutine which performs a transpose operation in Figure 3.2. The transpose of *A1* is stored in *TA*.

```
13320   REM     SUBROUTINE TO TRANSPOSE A MATRIX
13330   REM    INPUT MATRIX A1 TO BE TRANSPOSED
13340   REM    AND ITS DIMENSIONS, IROW AND ICOL
13350   REM    TRANSPOSE IS PLACED IN MATRIX TA
13360   DIM TA(ICOL,IROW)
13370   FOR I = 1 TO IROW
13380   FOR J = 1 TO ICOL
13390  TA(J,I) = A1(I,J)
13400   NEXT J,I
13410   RETURN
```

Figure 3.2—Matrix Transpose Subroutine

A main program which uses the transpose subroutine is given in Figure 3.3

```
100  REM  MAIN PROGRAM TO FIND THE TRANSPOSE OF A MATRIX
110  GOSUB 13010: REM          READ IN THE MATRIX
120  GOSUB 13320: REM          FIND THE TRANSPOSE, PLACE IN MATRIX TA
```

```
130  FOR I = 1 TO ICOL
140  FOR J = 1 TO IROW
150  PRINT TA(I,J);"    ";
160  NEXT J
170  PRINT
180  NEXT I
190  END
```

Figure 3.3—Using the Transpose Subroutine

EXAMPLE 1. Find the transpose of the following matrix:

$$A1 = \begin{bmatrix} 3 & 2 \\ 6 & 5 \end{bmatrix}$$

SOLUTION.

$$TA = \begin{bmatrix} 3 & 6 \\ 2 & 5 \end{bmatrix}$$

Addition and Subtraction

Matrices may be added or subtracted in a manner very similar to scalars. In fact, matrix addition consists of the addition of corresponding scalar elements.

Not all matrices can be added (or subtracted). For some pairs of matrices, the operation is undefined. In order for the operation to be defined, the matrices must be "conformable." This means that they conform to certain specifications which allow the operation to be performed correctly. Two matrices are "conformable" for addition if they have the same number of rows and columns. That is, the matrices must be the same "size."

Once it is determined that two matrices are conformable for addition, the actual addition is a straightforward process of scalar addition. We add scalars corresponding to identical row and column positions of both matrices to yield the new element in that row and column position in the sum matrix. In general,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

$$C = A+B = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

We present an example of this procedure below:

$$A = \begin{bmatrix} 2 & 5 & 2 \\ 0 & 7 & 9 \end{bmatrix} \qquad B = \begin{bmatrix} 6 & 10 & 12 \\ 0 & 1 & 1 \end{bmatrix}$$

$$C = A+B = \begin{bmatrix} 2+6 & 5+10 & 2+12 \\ 0+0 & 7+1 & 9+1 \end{bmatrix} = \begin{bmatrix} 8 & 15 & 14 \\ 0 & 8 & 10 \end{bmatrix}$$

Notice that C, the sum of A and B, will also have the same dimensions as A and B; m $\times$ n. Notice also that since addition is commutative for scalars, it is also commutative for matrices.

$$A + B = B + A$$

The rules for matrix subtraction are the same as those for addition, with minus signs instead of plus signs. Of course, subtraction is not commutative. Adding or subtracting matrices on the computer is easy. Figure 3.4 presents a series of subroutines for adding two matrices, *A1* and *B1*. A main program to calculate the sum of two matrices is presented in Figure 3.5.

```
13430   REM  SUBROUTINE ADD:
13440   REM   SUBROUTINE TO ADD TWO MATRICES, A1 AND B1, RESULT STORED IN C1
13450   REM   THEIR DIMENSIONS MUST BE IROW, ICOL; JROW, JCOL, RESPECTIVELY
13460   IERR = 1: REM          SET NO ERROR CONDITION
13470   IF IROW <  > JROW THEN IERR = 2: REM    SET ERROR CONDITION
13480   IF ICOL <  > JCOL THEN IERR = 2: REM    SET ERROR CONDITION
13490   ON IERR GOSUB 13570,13530
13500   RETURN
13510   REM ************************************************************
13520   REM  ERROR SUBROUTINE:
13530   REM   ERROR SUBROUTINE FOR NON-CONFORMABLE MATRICES
13540   PRINT : PRINT "MATRICES ARE NOT CONFORMABLE FOR THIS OPERATION"
13550   RETURN
13560   REM ************************************************************
13570   REM     MATRIX ADDITION SUBROUTINE
13580   DIM C1(IROW,ICOL)
13590   FOR I = 1 TO IROW
13600   FOR J = 1 TO ICOL
13610   C1(I,J) = A1(I,J) + B1(I,J)
13620   NEXT J,I
13630   RETURN
```

Figure 3.4—Add Subroutine

```
200  REM MAIN PROGRAM TO ADD TWO MATRICES
210  GOSUB 13130: REM         READ IN BOTH MATRICES
220  GOSUB 13440: REM         ADD THE MATRICES, STORE THE RESULT IN MATRIX C1
230  FOR I = 1 TO IROW
240  FOR J = 1 TO JCOL
250  PRINT C1(I,J);"  ";
260  NEXT J
270  PRINT
280  NEXT I
290  END
```

Figure 3.5—Using the Add Routine

EXAMPLE 2. Find the sum of

$$A1 = \begin{bmatrix} 6 & 2 \\ 5 & 3 \\ 1 & 2 \end{bmatrix} \text{ and } B1 = \begin{bmatrix} 7 & 7 \\ 1 & 0 \\ 3 & 0 \end{bmatrix}$$

SOLUTION.

$$C1 = A1 + B1 = \begin{bmatrix} 13 & 9 \\ 6 & 3 \\ 4 & 2 \end{bmatrix}$$

Multiplication by a Scalar

Matrices can also be used in multiplication. The simplest of these is multiplication by a scalar. All that is required in this instance is to multiply each element of the matrix by the scalar.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \qquad gA = \begin{bmatrix} ga_{11} & ga_{12} & \cdots & ga_{1n} \\ ga_{21} & ga_{22} & \cdots & ga_{2n} \\ \vdots & \vdots & & \vdots \\ ga_{m1} & ga_{m2} & \cdots & ga_{mn} \end{bmatrix}$$

This can be represented very easily in BASIC, as you can see by the subroutine in Figure 3.6.

```
13650   REM  MATRIX MULTIPLICATION (BY A SCALAR) ROUTINE:
13660   REM    SUBROUTINE TO MULTIPLY MATRIX A1 BY A SCALAR, S
13670   REM    THE RESULT IS STORED IN MATRIX D1. DIMENSIONS OF A1,
13680   REM    IROW AND ICOL, MUST BE PROVIDED.
13690   DIM D1(IROW,ICOL)
13700   FOR I = 1 TO IROW
13710   FOR J = 1 TO ICOL
13720   D1(I,J) = S * A1(I,J)
13730   NEXT J,I
13740   RETURN
```

Figure 3.6—Scalar, Matrix Multiplication Subroutine

Matrix Multiplication

Multiplication of matrices is a very important operation. Unfortunately, it is not as easy as matrix addition. Not all matrices may be multiplied together, and, matrix mul-

tiplication does not follow a commutative law. In general, $AB \neq BA$. We must specify the order of multiplication.

Two matrices A and B are "conformable" for the multiplication AB if the number of columns of A equals the number of rows of B. The resulting matrix will have the same number of rows as A and the same number of columns as B. In fact, the multiplication procedure itself matches the rows of A with the columns of B.

Let the product of A and B be C. We find the element in the i row and j column of C, called the ij-element, as follows. Consider only row i of A and column j of B. Since A and B are conformable for AB, row i of A and column j of B have the same number of elements. Sum the following terms: (first element of row i) $\times$ (first element of column j), (second element of row i) $\times$ (second element of column j) . . . (last element of row i) $\times$ (last element of column j). This sum is the ij element of C. We find each element of C in this manner. We illustrate this with the following example.

$$A = \begin{bmatrix} 6 & 1 \\ 7 & 3 \\ 2 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 7 \\ 6 & 3 \end{bmatrix}$$

$$\begin{aligned} C(1,1) &= (6)\,(1) + (1)\,(6) = 12 \\ C(1,2) &= (6)\,(7) + (1)\,(3) = 45 \\ C(2,1) &= (7)\,(1) + (3)\,(6) = 25 \\ C(2,2) &= (7)\,(7) + (3)\,(3) = 58 \\ C(3,1) &= (2)\,(1) + (4)\,(6) = 26 \\ C(3,2) &= (2)\,(7) + (4)\,(3) = 26 \end{aligned} \qquad C = \begin{bmatrix} 12 & 45 \\ 25 & 58 \\ 26 & 26 \end{bmatrix}$$

Note that the multiplication BA is not defined because the number of columns in B is not equal to the number of rows in A. Fortunately, while matrix multiplication is tedious if performed by hand, the computer makes it easy. In fact, reading a computer program for the procedure may help in understanding it. A set of BASIC subroutines for matrix multiplication is presented in Figure 3.7.

```
13760   REM  MATRIX MULTIPLICATION SUBROUTINE:
13770   REM   SUBROUTINE TO MULTIPLY MATRICES A1 AND B1 AS A1 TIMES B1
13780   REM   THE RESULT IS STORED IN MATRIX AB1. ROW AND COLUMN DIMENSIONS
13790   REM   OF A1 AND B1 MUST BE PROVIDED AS IROW, ICOL; JROW, JCOL, RESP.
13800   IERR = 1: REM            SET NO-ERROR CONDITION
13810   IF ICOL <  > JROW THEN IERR = 2: REM    SET ERROR CONDITION
13820   ON IERR GOSUB 13850, 13530
13830   RETURN
13840   REM ************************************************************
13850   REM    MAT MULT SUB
13860   REM   ACCESS ONLY THROUGH SUB 13760
13870   IF IXE <  > 0 THEN GOTO 13890: REM   SKIP DIMENSION FOR
        EXPONENTIATION
13880   DIM AB1(IROW,JCOL)
13890   FOR I = 1 TO IROW
13900   FOR J = 1 TO JCOL
13910   AB1(I,J) = 0
13920   FOR K = 1 TO ICOL
```

```
13930   AB1(I,J) = AB1(I,J) + A1(I,K) * B1(K,J)
13940   NEXT K,J,I
13950   RETURN
```

Figure 3.7—Matrix Multiplication Subroutine

A main program for multiplying two matrices, *A1* and *B1* is presented in Figure 3.8.

```
440  REM  MAIN PROGRAM TO MULTIPLY TWO MATRICES TOGETHER; A1 TIMES B1
450  GOSUB 13130: REM      READ IN BOTH MATRICES
460  GOSUB 13770: REM      MULTIPLY THE MATRICES, STORE RESULT IN MATRIX AB1
470  FOR I = 1 TO IROW
480  FOR J = 1 TO JCOL
490  PRINT AB1(I,J);"  ";
500  NEXT J
510  PRINT
520  NEXT I
530  END
```

Figure 3.8—Using Matrix Multiplication Subroutine

EXAMPLE 3. Find *A1* times *B1*,

$$A1 = \begin{bmatrix} 2 & 1 \\ 5 & 3 \\ 1 & 4 \end{bmatrix} \qquad B1 = \begin{bmatrix} 7 & 5 \\ 8 & 1 \end{bmatrix}$$

SOLUTION.

$$AB = A1 \times B1 = \begin{bmatrix} 22 & 11 \\ 59 & 28 \\ 39 & 9 \end{bmatrix}$$

Exponentiation

Quite often you may wish to multiply a matrix by itself a number of times. This process is matrix exponentiation. In order to perform this operation, or series of multiplications, the matrix must be square. Otherwise, it would not be conformable for multiplication with itself. A subroutine for this procedure is presented in Figure 3.9.

```
14680   REM  MATRIX EXPONENTIATION SUBROUTINE:
14690   REM     SUBROUTINE FOR MATRIX EXPONENTIATION.
14700   REM       IXE EQUALS THE POWER OF THE MATRIX DESIRED.
14710   REM     INPUT IS MATRIX A1 WITH DIMENSIONS IROW=ICOL
14720   REM     OUTPUT IS MATRIX C1 WITH THE SAME DIMENSIONS.
14730   DIM B1(IROW,ICOL)
14740   JROW = IROW
14750   JCOL = ICOL
14760   FOR I = 1 TO IROW
14770   FOR J = 1 TO ICOL
14780   B1(I,J) = A1(I,J): REM              SET THE MATRIX B1 EQUAL TO A1
```

```
14790   NEXT J,I
14800   DIM AB1(IROW,ICOL): REM          SET DIMENSION FOR PRODUCT MATRIX
14810   FOR IK = 1 TO IXE - 1
14820   GOSUB 13770: REM                  MULTIPLY A1 TIMES B1
14830   FOR I = 1 TO IROW
14840   FOR J = 1 TO ICOL
14850   B1(I,J) = AB1(I,J): REM          PLACE THE PRODUCT MATRIX BACK IN B1
14860   NEXT J,I
14870   NEXT IK
14880- RETURN
```

Figure 3.9—Matrix Exponentiation Subroutine

We illustrate a main program for matrix exponentiation in Figure 3.10.

```
810   REM   MAIN PROGRAM FOR MATRIX EXPONENTIATION
820   INPUT "INPUT THE POWER OF THE MATRIX DESIRED ?";IXE
830   GOSUB 13010: REM                  READ IN THE MATRIX A1
840   GOSUB 14680: REM                  PERFORM THE EXPONENTIATION
850   PRINT "FOR A1 TO THE ";IXE;" THE RESULT IS :"
860   FOR I = 1 TO IROW
870   FOR J = 1 TO JCOL
880   PRINT AB1(I,J);"   ";
890   NEXT J
900   PRINT
910   NEXT I
920   END
```

Figure 3.10—Using the Matrix Exponentiation Subroutine

EXAMPLE 4. Find $A1$ to the third power, where

$$A1 = \begin{bmatrix} 4 & 0 & 3 \\ 1 & 1 & 0 \\ 0 & 6 & 2 \end{bmatrix}$$

SOLUTION.

$$A1 = \begin{bmatrix} 82 & 126 & 84 \\ 21 & 19 & 21 \\ 42 & 42 & 26 \end{bmatrix}$$

Determinant

The determinant of a matrix is a scalar that summarizes certain information about the matrix. It is an operation involving only one matrix. Determinants exist only for square matrices. There are several methods we could use to find the determinant of a matrix. However, one is more easily programmed than the others, so that one will be used. We will use a technique called Gaussian elimination.

Gaussian elimination is a matrix reduction technique. We will use it to obtain a matrix that is upper triangular. In an upper triangular matrix, elements below and to the left of

the diagonal are zero. This upper triangular matrix will be significantly different from our original matrix, but it will have the same determinant. Matrix A below is an upper triangular matrix.

$$A = \begin{bmatrix} 7 & 2 & 9 & 2 \\ 0 & 5 & 10 & 6 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The determinant is found as the product of the diagonal elements. We will also see the usefulness of Gaussian elimination when we find the inverse of a matrix and solve simultaneous linear equations.

Gaussian elimination consists of the repeated application of row operations to a matrix. A row operation is the addition of a scalar multiple of one row to another row. We must try to ensure that the diagonal elements of the matrix are not zero. This may require additional row operations. If the diagonal elements cannot all be made nonzero, then the determinant is zero.

Using row operations, we must force the appropriate elements of the matrix to zero. The following example illustrates this procedure:

$$B = \begin{bmatrix} 1 & 5 & 1 \\ 3 & 6 & 3 \\ 0 & 1 & -9 \end{bmatrix}$$

1. Multiply the top row by -3 and add to the second row

$$B_1 = \begin{bmatrix} 1 & 5 & 1 \\ 0 & -9 & 0 \\ 0 & 1 & -9 \end{bmatrix}$$

2. Multiply the second row by $\frac{1}{9}$ and add to the third row

$$B_2 = \begin{bmatrix} 1 & 5 & 1 \\ 0 & -9 & 0 \\ 0 & 0 & -9 \end{bmatrix}$$

Matrix B_2 is upper triangular, so we multiply the diagonal elements to find the determinant. This will be the same as the determinant of B.

$$\text{Determinant of } B = |B| = |B_2| = (1)(-9)(-9) = 81$$

Figure 3.11 presents BASIC subroutines to create an upper triangular matrix, and thus find the determinant of the original matrix.

```
13970   REM  UPPER TRIANGULAR SUBROUTINE:
13980   REM   SUBROUTINE TO CREATE AN UPPER TRIANGULAR MATRIX
13990   REM   PROVIDE MATRIX A1 AND ITS DIMENSIONS, IROW=ICOL. RESULT IN E1
14000   IERR = 1: REM       SET NO-ERROR CONDITION
14010   IF IROW <  > ICOL THEN IERR = 2: REM    SET ERROR CONDITION
14020   ON IERR GOSUB 14060, 13530
14030   RETURN
14040   REM ************************************************************
14050   REM  UPPER TRI ROUTINE
14060   REM   UPPER TRIANGULAR ROUTINE
14070   REM   ACCESS ONLY THROUGH SUB 13970
14080   IF INV = 1 THEN ICOL = 2 * ICOL: REM    SET PARAMETER FOR MATRIX
        INVERSE
14100   DIM E1(IROW,ICOL)
14110   FOR I = 1 TO IROW
14120   FOR J = 1 TO IROW
14130   E1(I,J) = A1(I,J): REM    ASSIGN E1 MATRIX TO A1
14140   IF INV <  > 1 THEN   GOTO 14180: REM    SKIP IDENTITY AUGMENTATION
14150   REM   THIS SECTION IS USED ONLY IF A MATRIX IS BEING INVERTED
14160   IF I = J THEN E1(I,J + IROW) = 1: REM    AUGMENT THE IDENTITY MATRIX
        TO E1
14180   NEXT J
14200   NEXT I
14210   DET = 1: REM    SET DETERMINANT FLAG
14220   FOR I = 1 TO IROW - 1
14230   IF E1(I,I) = 0 THEN   GOSUB 14330: REM    DIAGONAL ELEMENT MUST NOT BE
        ZERO
14240   IF DET = 0 THEN   RETURN
14250   FOR J = I + 1 TO IROW
14260   XM = E1(J,I) / E1(I,I): REM   MULTIPLIER TO ZERO THE COLUMN ELEMENTS
14270   FOR K = I TO ICOL
14280   E1(J,K) = E1(J,K) - XM * E1(I,K): REM    CALCULATE NEW ELEMENTS
14290   NEXT K,J,I
14300   RETURN
14310   REM ************************************************************
14320   REM   ZERO DETERMINANT DETECTION SUBROUTINE:
14330   REM    SUBROUTINE TO ENSURE DIAGONAL ELEMENTS ARE NONZERO DURING
14340   REM    UPPER-TRIANGULARIZATION. IF THIS IS NOT POSSIBLE, DET IS ZERO
14350   FOR J = I + 1 TO IROW
14360   IF E1(J,I) = 0 THEN   GOTO 14410
14370   FOR K = 1 TO ICOL
14380   E1(I,K) = E1(I,K) + E1(J,K): REM    ADD ROWS TO MAKE DIAGONAL NONZERO
14390   NEXT K
14400   RETURN
14410   NEXT J
14420   DET = 0; REM              DET MUST BE ZERO AT THIS POINT
14430   RETURN
```

Figure 3.11—Upper Triangular Matrix Subroutine

A main program for the entire procedure is in Figure 3.12.

```
550   REM   MAIN PROGRAM TO FIND THE DETERMINANT OF A MATRIX
560   GOSUB 13010: REM                READ IN THE MATRIX
570   GOSUB 13970: REM                FIND THE DETERMINANT
580   IF DET = 0 THEN   GOTO 620
590   FOR I = 1 TO IROW
```

```
600   DET = DET * E1(I,I)
610   NEXT I
620   PRINT "DETERMINANT IS EQUAL TO ";DET
630   END
```

Figure 3.12—Using Upper Triangular Matrix Subroutine

EXAMPLE 5. Find the determinant of:

$$AI = \begin{bmatrix} 3 & 7 & 5 \\ 9 & 0 & 1 \\ 0 & 4 & 3 \end{bmatrix}$$

SOLUTION.

$$|AI| = -21$$

Inverse

The inverse of a matrix serves the same purpose as the reciprocal for scalars. This is the only way a sense of division is possible for matrices. We can find inverses for square matrices only.

In order to define the inverse of a matrix, we must first define the identity matrix. An identity matrix is a square matrix with ones along its diagonal and zeros elsewhere. It is generally assigned the letter I. The following are identity matrices.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Given any matrix A, the inverse of A, denoted A^{-1}, is a matrix such that:

$$AA^{-1} = A^{-1}A = I$$

Of course, I must have the same dimensions as A. We present the matrix B below along with its inverse. Notice that their product is I.

$$B = \begin{bmatrix} 1 & 3 \\ 2 & 8 \end{bmatrix} \qquad B^{-1} = \begin{bmatrix} 4 & -3/2 \\ -1 & 1/2 \end{bmatrix}$$

We can find the inverse of a matrix in many ways, but we consider only the Gaussian elimination method here. We can build upon the knowledge developed in the last section. If you have not read the section on determinants yet, we suggest you do so prior to the development below.

To find the inverse, we want to perform row operations on our matrix which will turn it into an identity matrix. If we perform the same row operations on an identity matrix,

the result will be the inverse of our original matrix. Therefore, we will perform the row operations required on the original matrix and the identity matrix simultaneously. When the original matrix becomes an identity, its inverse is found.

The inverse of a square matrix does not always exist. This occurs if the determinant of the matrix is zero. It is beneficial to check this condition as early as possible to avoid unnecessary computations. If the determinant is zero, then the matrix is called "singular."

We will begin by augmenting the matrix with an identity by placing the identity beside the original matrix. This is illustrated below for finding B^{-1}.

$$B = \begin{bmatrix} 1 & 3 \\ 2 & 8 \end{bmatrix} \qquad B|I = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 8 & 0 & 1 \end{array}\right]$$

We now want to perform row operations to transform B into an identity. To do this we will first make it upper triangular, then make it lower triangular. In a lower triangular matrix, elements above and to the right of the diagnonal are zero. Thus, we are left with a matrix that has only diagonal elements which may be nonzero. In addition to adding a scalar multiple of a row to another, we can also simply multiply a row by a scalar. All operations are performed over the entire $B|I$ matrix. These operations will not change the inverse. Lets see how this works for this example.

1. Multiply the first row by -2 and add to the second row

$$B|I_1 = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & 2 & -2 & 1 \end{array}\right]$$

At this stage we see that the determinant of B is not equal to zero, so the inverse will exist.

2. Multiply the second row by $-3/2$ and add to the first row.

$$B|I_2 = \left[\begin{array}{cc|cc} 1 & 0 & 4 & -3/2 \\ 0 & 2 & -2 & 1 \end{array}\right]$$

3. Multiply the second row by 1/2.

$$B|I_3 = \left[\begin{array}{cc|cc} 1 & 0 & 4 & -3/2 \\ 0 & 1 & -1 & 1/2 \end{array}\right]$$

We see that the original matrix is now an identity matrix and the original identity matrix has been replaced by B^{-1}. Therefore,

$$B^{-1} = \begin{bmatrix} 4 & -3/2 \\ -1 & 1/2 \end{bmatrix}$$

A subroutine to form a lower triangular matrix is illustrated in Figure 3.13.

```
14450   REM   LOWER TRIANGULARIZATION SUBROUTINE:
14460   REM   SUBROUTINE TO MAKE A MATRIX LOWER TRIANGULAR, THEN AN IDENTITY.
14470   REM   FIND THE MULTIPLIER TO ZERO ELEMENTS IN THE KTH COLUMN
14480   REM   ABOVE THE DIAGONAL.
14490   REM   INPUT IS THE MATRIX E1 WITH ITS DIMENSIONS IROW AND ICOL
14500   FOR IJ = 1 TO IROW
14510   IF IJ = IROW THEN   GOTO 14580
14520   IK = IROW - IJ + 1
14530   FOR I = 1 TO IK - 1
14540  XM = E1(I,IK) / E1(IK,IK)
14550   FOR J = I + 1 TO ICOL
14560  E1(I,J) = E1(I,J) - XM * E1(IK,J)
14570   NEXT J,I
14580   NEXT IJ
14590   REM CREATE IDENTITY BY MULTIPLYING EACH ROW BY THE RECIPROCAL OF THE
14600   REM DIAGONAL ELEMENT OF THE REVISED MATRIX
14610   FOR I = 1 TO IROW
14620  DIV = E1(I,I)
14630   FOR J = 1 TO ICOL
14640  E1(I,J) = E1(I,J) / DIV
14650   NEXT J,I
14660   RETURN
```

Figure 3.13—Lower Triangular Matrix Subroutine

Putting the appropriate subroutines together, the inverse of a matrix can be found using the main program in Figure 3.14.

```
650  REM   MAIN PROGRAM TO FIND THE INVERSE OF A MATRIX
670  INV=1                              'Set inverse flag
680  GOSUB 13010                        'Read in the matrix
690  GOSUB 13980                        'Make the matrix upper triangular
700  IF DET=0 THEN PRINT "MATRIX IS SINGULAR"
710  IF DET=0 THEN GOTO 800
720  GOSUB 14450                        'Make E1 lower triangular
730  PRINT "THE INVERSE OF A1 IS"
740  FOR I=1 TO IROW
750  FOR J=IROW+1 TO ICOL
760  PRINT E1(I,J);
770  NEXT J
780  PRINT
790  NEXT I
800  END
```

Figure 3.14—Matrix Inverse Program

EXAMPLE 6. Find the inverse of:

$$A1 = \begin{bmatrix} 3 & 1 & 6 \\ 0 & 2 & 5 \\ 1 & 1 & 1 \end{bmatrix}$$

SOLUTION.

$$AI^{-1} = \begin{bmatrix} 0.1875 & -0.3125 & 0.4375 \\ -0.3125 & 0.1875 & 0.9375 \\ 0.125 & 0.125 & -0.375 \end{bmatrix}$$

3.3 Vectors

We will present no special programs for vectors. They can be treated as matrices with only one column. An example of a vector is:

$$\vec{a} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

Here, $\vec{a}$ is a three-element column vector, or a 3×1 matrix. All of the matrix operations we discussed earlier are also appropriate for vectors. Remember that since a vector can never be square, it cannot be multiplied by itself directly and it does not have a determinant or an inverse.

Since a vector is still an array, it must be defined in the DIM statement. For $\vec{b}$ a five-element vector and $\vec{c}$ a twenty-element vector, we need the following DIM statement:

```
DIM B(5), C(20)
```

3.4 Summary

In this chapter we have presented many commonly used matrix and vector operations. These operations may come in quite handy when you are confronted with large problems. The subroutines presented may be used in combination or separately, depending on the situation. You will want to keep an eye on the "condition" of your problem, an issue which is quite important but which is not addressed here. For example, the true solution to a problem may be f(x), but using the computer, and possibly numerical techniques, yields an approximation to the solution f(x*). If f(x) and f(x*) differ greatly, then the problem is "ill-conditioned." Stewart (1973) presents an excellent discussion of this topic. Based upon further study in this area, you might want to revise some of the programs here to handle specific conditioning problems. You will see the value of the techniques presented here when examining solutions to simultaneous linear equations (Chapter 5) and linear programming (Chapter 9).

References

Hadley, G *Linear Algebra*, Addison-Wesley Publishing Company, Inc., Reading, Mass. 1961.

Stewart, G. W. *Introduction to Matrix Computations*, Academic Press, Inc., New York, N.Y., 1973.

Exercises

1. Consider the following two matrices:

$$A = \begin{bmatrix} 3 & 5 & 1 \\ 6 & 0 & 2 \\ 1 & 9 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 9 & 4 & 1 \\ 0 & 0 & 6 \\ 5 & 3 & 4 \end{bmatrix}$$

   Use these to illustrate that $AB \neq BA$. Develop two matrices for which $AB = BA$.
2. Write a computer program to verify the associative law for matrix multiplication: $(AB)C = A(BC)$. Be sure matrices are conformable for all necessary operations.
3. The set of simultaneous equations below:

$$\begin{aligned} 3x_1 + 6x_2 + 7x_3 &= b_1 \\ 4x_1 + 2x_2 + x_3 &= b_2 \\ x_2 - x_3 &= b_3 \end{aligned}$$

   can be written in matrix and vector notation as

$$\begin{bmatrix} 3 & 6 & 7 \\ 4 & 2 & 1 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

   For two different sets of $\{x_1, x_2, x_3\}$ find the values of $\{b_1, b_2, b_3\}$.
4. Create a square matrix A whose determinant is zero and multiply it by a square matrix B whose determinant is not zero. What is the determinant of the resulting matrix?
5. Write a computer program to illustrate that $(AB)^T = B^T A^T$.
6. Write a computer program to illustrate that $(AB)^{-1} = B^{-1} A^{-1}$.
7. What is the relationship between $|A|$ and $|kA|$, where k is any scalar?
8. Find the inverse of the following matrix.

$$A = \begin{bmatrix} 6 & 0 & 9 & 1 \\ 3 & 2 & 0 & 2 \\ 1 & 1 & 0 & 1 \\ 7 & 4 & 3 & 8 \end{bmatrix}$$

   Verify that you found the inverse.

4

Curve Fitting With Linear Regression

4.1 Linear Regression

Linear regression is the most widely used curve fitting technique. It is based on fitting (drawing) a straight line through a series of observed data points so that the deviations of these points from this line are minimized. A straight line can be expressed as $Y = A + BX$, where A is the intercept and B is the slope. Using linear regression, we can calculate values for A and B. Then, using some value for X (the independent variable), a value $\hat{Y}$ (the dependent variable) can be calculated that resides on the regression line.

Because most data will contain random variations, a particular observed data point, Y, may not reside on the regression line. One way to measure the amount of deviation about a regression line is to calculate the error associated with each point:

$$E_t = Y_t - \hat{Y}_t, \tag{1}$$

where Y_t is the observed data point and $\hat{Y}_t$ is the corresponding point on the fitted line. Figure 4.1 illustrates a regression line and the deviation associated with a dependent variable, Y.

Regression is based on determining the line that will minimize the sum of squares of the deviations. Thus, we want to minimize

$$\sum_{t=1}^{n} E_t^2 = \sum_{t=1}^{n} [Y_t - \hat{Y}_t]^2 \tag{2}$$

Since Y can be expressed in terms of an equation for a line, we get:

$$\sum_{t=1}^{n} E_t^2 = \sum_{t=1}^{n} [Y_t - (A + BX_t)]^2 \tag{3}$$

The problem is to choose the coefficients A and B in equation 3 such that the sum of the errors squared is minimized. We can do this using differential calculus. Differentiating equation 3 with respect to A and B, we get:

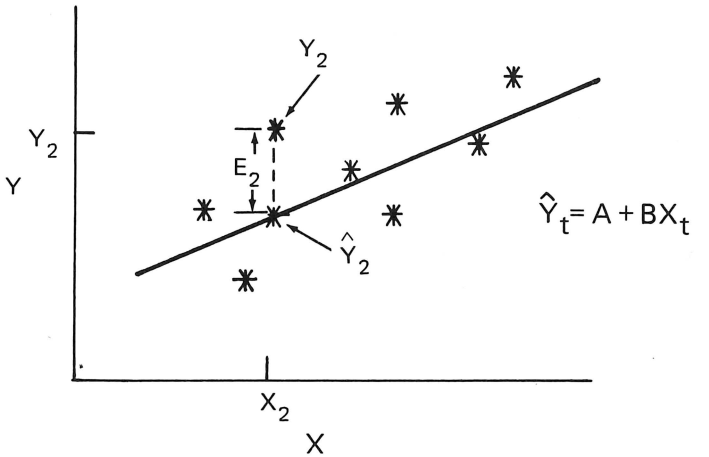


Figure 4.1—Deviation From a Regression Line

$$\frac{\delta\left[\sum_{t=1}^{n} E_t^2\right]}{\delta A} = -2\sum_{t=1}^{n}\left[Y_t - (A+BX_t)\right] \tag{4}$$

$$\frac{\delta\left[\sum_{t=1}^{n} E_t^2\right]}{\delta B} = -2\sum_{t=1}^{n}\left[Y_t X_t - (AX_t+BX_t^2)\right] \tag{5}$$

When equations 4 and 5 are equated to 0, we get two simultaneous equations with two unknowns. Solving these equations gives:

$$A = \frac{\sum_{t=1}^{n} X_t^2 \sum_{t=1}^{n} Y_t - \left[\sum_{t=1}^{n} X_t \sum_{t=1}^{n} X_t Y_t\right]/n}{\sum_{t=1}^{n} X_t^2 - \sum_{t=1}^{n} X_t \sum_{t=1}^{n} X_t/n} \tag{6}$$

$$B = \frac{\sum_{t=1}^{n} X_t Y_t - \sum_{t=1}^{n} X_t \sum_{t=1}^{n} Y_t/n}{\sum_{t=1}^{n} X_t^2 - \sum_{t=1}^{n} X_t \sum_{t=1}^{n} X_t/n} \tag{7}$$

If these equations must be solved by hand, they appear formidable. However, these computations can be done easily by a computer.

After fitting a line through a set of data, we need some way to determine if the fit is very good. One indicator is the *coefficient of correlation*, which can have a value between -1 and $+1$. It may be expressed as

$$CC = \frac{n\sum_{t=1}^{n} X_t Y_t - \sum_{t=1}^{n} X_t \sum_{t=1}^{n} Y_t}{\sqrt{\left[n\sum_{t=1}^{n} X_t^2 - \left(\sum_{t=1}^{n} X_t\right)^2\right]\left[n\sum_{t=1}^{n} Y_t^2 - \left(\sum_{t=1}^{n} Y_t\right)^2\right]}} \tag{8}$$

A positive coefficient of correlation indicates that increases in the independent variable will result in increases in the dependent variable. The variables are directly related. If the coefficient of correlation is negative, the variables are inversely related; that is, an increase in the independent variable causes a decrease in the dependent variable. A coefficient of correlation of 0 means that the variables are not related.

Another useful indicator is the *coefficient of determination*. It is equal to the square of the coefficient of correlation:

$$CD = CC^2. \tag{9}$$

One interpretation of this indicator is that it denotes the percentage of change in the dependent variable that can be explained by changes in the regression line. A coefficient of determination of 1.0 means that 100% of the changes are explained. Consequently, this coefficient will have a value between 0 and 1.

Another useful statistic is the standard error of the estimate:

$$SE = \sum_{t=1}^{n} (Y_t - \hat{Y}_t)^2/(n-2) \tag{10}$$

It is a measure of how closely the predicted values, Y, compare with the actual values, Y. It is desirable to have this value as small as possible.

The quantity, $\sum_{t=1}^{n} (Y_t - \hat{Y}_t)^2$, in Equation 10 is known as the sum of squares about the regression. This value can be expressed in another way:

$$\sum_{t=1}^{n} (Y_t - \hat{Y}_t)^2 = \sum_{t=1}^{n} Y_t - \left(\sum_{t=1}^{n} Y_t\right)^2/n - B\left[\sum_{t=1}^{n} X_t Y_t - \left(\sum_{t=1}^{n} X_t \sum_{t=1}^{n} Y_t\right)/n\right] \tag{11}$$

(The interested reader might refer to reference [1] to see how this expression is derived.) Although the expression on the right side of the equal sign appears complicated, it is easy to implement in a computer program. In fact, it is preferable to the expression on the left because all of the terms (Y_t, $X_t Y_t$, etc.) have been previously calculated in the process of obtaining values for A and B.

Figure 4.2 contains a program that can be used to perform linear regression where the regression line has the form

$$Y = A + BX.$$

The program in Figure 4.2 is not a subroutine, although it contains some subroutines. Consequently, the user does not have to write a main program to use this program.

```
1000  PRINT : PRINT  TAB( 8);"SIMPLE LINEAR REGRESSION"
1010  PRINT  TAB( 7);"==========================": PRINT
1020  REM   USING LINEAR REGRESSION THIS PROGRAM WILL ESTIMATE A
1030  REM   LINE, Y=A+BX, WHERE X IS THE INDEPENDENT VARIABLE AND
1040  REM   Y IS THE DEPENDENT VARIABLE. IF MORE THAN 30
1050  REM   OBSERVATIONS ARE USED, THE DIMENSION STATEMENTS MUST
1060  REM   BE CHANGED. SUBROUTINE REGRESSION MAY BE USED BY OTHER
1070  REM   PROGRAMS IF DATA IS PROVIDED IN THE ARRAYS X AND Y AND
1080  REM   THE NUMBER OF OBSERVATIONS IS PROVIDED IN VARIABLE IN.
1090  REM
1100  DIM X(30),Y(30)
1110  PRINT : INPUT "INPUT NUMBER OF DATA POINTS ? ";IN
1120  IF IN <  = 0 THEN 1110
1130  IF IN < 4 THEN  PRINT "NO. MUST BE > 3": GOTO 1110
1140  PRINT : PRINT "INPUT DATA IN PAIRS : X,Y"
1150  PRINT "WHERE X IS INDEPENDENT VARIABLE AND Y IS DEPENDENT VARIABLE": PRINT
1160  FOR I = 1 TO IN
1170  PRINT "INPUT X,Y FOR POINT ";I;
1180  INPUT X(I),Y(I)
1190  NEXT I
1200  PRINT : PRINT "AVAILABLE OPTIONS :": PRINT
1210  PRINT  TAB( 7)"1-LIST INPUT DATA"
1220  PRINT  TAB( 7)"2-MODIFY INPUT DATA"
1230  PRINT  TAB( 7)"3-PERFORM REGRESSION ANALYSIS"
1240  PRINT  TAB( 7)"4-QUIT"
1250  PRINT : INPUT "OPTION ? ";IP
1260  IF (IP < 1) OR (IP > 4) THEN 1200
1270  IF IP = 1 THEN  GOSUB 1340
1280  IF IP = 2 THEN  GOSUB 1470
1290  IF IP = 3 THEN  GOSUB 1550
1300  IF IP = 4 THEN 1990
1310  GOTO 1200
1320  REM  ******************************************************
1330  REM  SUBROUTINE: LIST DATA
1340  PRINT : PRINT "LISTING OF DATA :": PRINT
1350  PRINT "   X","   Y"
1360  PRINT "-------","-------"
1370 IC = 1
1380  FOR I = 1 TO IN
1390  IF I <  > (IC * 15) THEN 1420
1400 IC = IC + 1
1410  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
1420  PRINT  TAB( 2);X(I); TAB( 18);Y(I)
1430  NEXT I
1440  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: RETURN
1450  REM  ******************************************************
1460  REM  SUBROUTINE: MODIFY DATA
1470  PRINT : INPUT "ENTER # OF DATA POINT TO BE MODIFIED ? ";ID
1480  PRINT "NEW VALUES FOR X AND Y FOR POINT ";ID;" ";
1490  INPUT X(ID),Y(ID)
1500  PRINT "ANY MORE DATA POINTS TO BE MODIFIED"
1510  INPUT "(Y/N) ? ";Y$
1520  IF Y$ = "Y" THEN 1470
1530  RETURN
1540  REM  ******************************************************
1550  REM  SUBROUTINE: REGRESSION
1560 SX = 0:SY = 0:X2S = 0:Y2S = 0:XYS = 0
1570  FOR I = 1 TO IN
1580 SX = SX + X(I): REM      SUM OF X
1590 SY = SY + Y(I): REM      SUM OF Y
1600 X2S = X2S + X(I) ^ 2: REM  SUM OF X^2
1610 Y2S = Y2S + Y(I) ^ 2: REM  SUM OF Y^2
1620 XYS = XYS + X(I) * Y(I): REM  SUM OF X*Y
1630  NEXT I
1640 B = (IN * XYS - SX * SY) / (IN * X2S - SX ^ 2): REM   SLOPE OF LINE
1650 A = (SY - B * SX) / IN: REM   INTERCEPT OF LINE
```

```
1660  REM  COEFFICIENT OF CORRELATION
1670 CC = (XYS - SX * SY / IN) / ( SQR ((X2S - (SX ^ 2) / IN) * (Y2S - (SY ^ 2) / IN)))
1680 CR = CC ^ 2: REM   COEFFICIENT OF DETERMINATION
1690 SSE = Y2S - SY ^ 2 / IN - B * (XYS - SX * SY / IN): REM   ERROR SUM OF SQUARES
1700 SE =  SQR (SSE / (IN - 2)): REM   STD. DEVIATION OF ESTIMATE
1710  REM  ************************************************************
1720  REM  SUBROUTINE: PRINT RESULTS
1730 A =  INT (A * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
1740 B =  INT (B * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
1750  PRINT : PRINT "REGRESSION EQUATION : ";
1760  PRINT "Y=";A;"+";B;"X": PRINT
1770 CR =  INT (CR * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
1780 CC =  INT (CC * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
1790 SE =  INT (SE * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
1800  PRINT "COEFFICIENT OF DETERMINATION=";CR
1810  PRINT "COEFFICIENT OF CORRELATION=";CC
1820  PRINT "STANDARD DEVIATION OF THE ESTIMATE=";SE
1830  PRINT : PRINT "ACTUAL VERSUS ESTIMATED VALUES"
1840  PRINT : PRINT "  X        Y      ESTIMATED Y      ERROR"
1850  PRINT "------   ------   ------------    --------"
1860 IC = 1
1870  FOR I = 1 TO IN
1880  IF I <  > (IC * 15) THEN 1910
1890  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
1900 IC = IC + 1
1910 EY = A + B * X(I)
1920 ER = Y(I) - EY
1930 ER =  INT (ER * 10 ^ 3 + .5) /  INT (10 ^ 3 + .5)
1940 EY =  INT (EY * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
1950  PRINT  TAB( 2);X(I); TAB( 10);Y(I); TAB( 18);EY; TAB( 31);ER
1960  NEXT I
1970  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
1980  PRINT : RETURN
1990  PRINT : PRINT  TAB( 14)"END OF PROGRAM"
```

Figure 4.2—Linear Regression Program

However, the REGRESSION subroutine starting at line number 1520 can be used by another program if data are provided in the arrays X and Y and the number of pairs of data is provided in the variable IN.

This regression program prompts you for the values for X and Y; as many as 30 pairs of data may be input. If you have more than 30 pairs of values, the dimension statement in the program for the variables X and Y must be changed. After you have input your data, you are given the opportunity to list the data, correct the data, perform the regression analysis, or quit. If you select the regression analysis option, the parameters (A and B) of the regression line, the coefficient of correlation, the coefficient of determination, and the standard error of the estimate are computed and displayed. Lastly, the input data are displayed with the respective computed values, $\hat{Y}$; and the error of each estimated.

EXAMPLE 1. A company, known for high quality products, has had to dispose of unusually large amounts of products that did not pass specifications. In searching for a solution to this problem, it has been noted that the impurities in chemical A, which is used in these products, have changed. The following data have been collected:

Month	Pounds of Product Rejected/Month (Thousands of Pounds)	Average Amount of Impurities in Chemical A (Parts per Million)
1	115	10
2	249	31
3	208	17
4	374	42
5	307	36
6	299	33

From the data, develop an equation which can be used to predict the pounds of product that will be rejected if the average impurity of chemical A is known.

SOLUTION. The linear regression program in Figure 4.2 can be used to determine if a regression line with a high coefficient of determination can be obtained using the compiled data. The results of the analysis are shown in Figure 4.3. Using these results, the amount of product rejected can be predicted using the following expression:

$$Y = 55.95308 + 7.19693(X)$$

It appears that there is a correlation between the impurities of chemical A and the amount of product rejected.

```
LOAD SIMREG
RUN

    SIMPLE LINEAR REGRESSION
    ========================

INPUT NUMBER OF DATA POINTS ? 6

INPUT DATA IN PAIRS : X,Y
WHERE X IS INDEPENDENT VARIABLE, AND
Y IS DEPENDENT VARIABLE

INPUT X,Y FOR POINT 1?10,115
INPUT X,Y FOR POINT 2?31,249
INPUT X,Y FOR POINT 3?17,208
INPUT X,Y FOR POINT 4?42,374
INPUT X,Y FOR POINT 5?36,307
INPUT X,Y FOR POINT 6?33,299

AVAILABLE OPTIONS :

      1-LIST INPUT DATA
      2-MODIFY INPUT DATA
      3-PERFORM REGRESSION ANALYSIS
      4-QUIT

OPTION ? 3

REGRESSION EQUATION : Y=55.95308+7.19693X
```

```
COEFFICIENT OF DETERMINATION = .94338
COEFFICIENT OF CORRELATION = .97126
STANDARD DEVIATION OF THE ESTIMATE = 23.96271

ACTUAL VERSUS ESTIMATED VALUES

 X          Y          ESTIMATED Y          ERROR
 10         115         127.92238          -12.922
 31         249         279.05791          -30.058
 17         208         178.30089           29.699
 42         374         358.22414           15.776
 36         307         315.0426            - 8.043
 33         299         293.45177            5.548

PRESS RETURN TO CONTINUE ->

AVAILABLE OPTIONS:

      1-LIST INPUT DATA
      2-MODIFY INPUT DATA
      3-PERFORM REGRESSION ANALYSIS
      4-QUIT

OPTION ? 4

      END PROGRAM
```

Figure 4.3—Product Rejection Example

4.2 Multiple Regression

The preceding discussion involved equations with one independent and one dependent variable:

$$Y = A + BX.$$

This approach may be extended to include several independent variables:

$$Y = A + B_1X_1 + B_2X_2 + \ldots + B_nX_n. \tag{12}$$

If more than one independent variable is involved, this technique is called multiple regression.

It is possible to apply the regression approach to an equation of the form:

$$Y = A + B_1Z + B_2Z_1^2 + B_3Z_1Z_2 + \ldots B_nf_n(Z_1, Z_2 \ldots, Z_n) \tag{13}$$

This equation can be made equivalent to (12) by the following substitutions:

$$X_1 = Z_1$$
$$X_2 = Z_1^2$$
$$X_3 = Z_1Z_2$$
$$\bullet$$
$$\bullet$$
$$\bullet$$
$$X_n = f_n(Z_1, Z_2 \ldots, Z_n)$$

When the above substitutions have been made, we say that equation (13) has been "transformed" to a linear model.

Regression can be applied to other nonlinear expressions if the proper transformations are first applied. For instance, consider

$$Y = Ae^{Bt} \tag{14}$$

A logarithmic transformation yields:

$$\ln Y = \ln A + Bt \tag{15}$$

The expression on the right represents the equation of a straight line where the intercept is ln A and the slope is B. In order to apply regression to expressions such as equation (13), the original data must first be transformed. A line is then fit to the transformed data using regression. After values are obtained for the intercept (ln A) and slope (B), ln Y can be calculated. Then true values for Y can be obtained by taking the anti-ln of Y.

4.3 Stepwise Regression

Stepwise multiple regression is a form of multiple regression that adds one variable at a time to the "best fit" regression equation. Thus, we might get:

$$Y = A + B_1X_1$$

$$Y = A + B_1'X_1 + B_2'X_2$$

$$Y = A + B_1''X_1 + B_2''X_2 + B_3''X_3$$

• •

• •

• •

If there are n potential independent variables and we know that all n variables will be in the best fit equation, then we would not want to use a stepwise regression procedure. However, often the problem is to determine which of the independent variables should be included in the regression equation. This can be complicated by the fact that some of the independent variables may be highly correlated (that is, not independent). One possible approach is to investigate all possible combinations of a set of independent variables. This approach rapidly becomes computationally unattractive because of the many combinations of variables involved. The stepwise regression procedure is an answer to this problem.

Using a stepwise regression procedure, variables are added in the order which makes the greatest improvement in the goodness of fit (reduction of the variance). In an early stage, a variable may enter the regression equation; however, as other variables are added, the initial variable may be removed from the equation if its contribution is indicated to be insignificant. Consequently, the final regression equation will only include statistically significant variables.

Figure 4.4 illustrates a program that utilizes the stepwise linear regression procedure. We will not describe the mathematical details of this program here; these details are in

```
1000  REM  MULTIPLE AND STEPWISE REGRESSION PROGRAM
1010  REM  THESE PROGRAMS WILL PERFORM MULTIPLE REGRESSION OR STEPWISE
1020  REM  REGRESSION.SOME DATA TRANSFORMATIONS ARE PROVIDED.DATA
1030  REM  MANAGEMENT SUBROUTINES ARE INCLUDED. A FORECAST CAN BE MADE
1040  REM  USING THE ESTIMATED REGRESSION EQUATION. RESULTS MAY BE
1050  REM  PRINTED AT CRT OR PRINTER.AS MANY AS 11 VARIABLES(10
1060  REM  INDEPENDENT AND 1 DEPENDENT)CAN BE CONSIDERED WITH EACH
1070  REM  VARIABLE HAVING 30 OBSERVATIONS. A LARGER PROBLEM CAN BE
1080  REM  STUDIED BY CHANGING THE DIMENSION STATEMENTS.
1090  REM       IT=TOTAL NUMBER OF VARIABLES IN PROBLEM
1100  REM       IV=NUMBER OF INDEPENDENT VARIABLES
1110  REM       IO=NUMBER OF DATA POINTS INCLUDING FORECAST
1120  REM       IH=NUMBER OF DATA POINTS NOT INCLUDING FORECAST
1130  REM       VNAM$(I)=ARRAY CONTAINING NAMES OF VARIABLES
1140  REM       X(I,J)=OBSEVED DATA
1150  REM       Y(I,J)=ESTIMATED VALUES FOR DEPENDENT VARIABLE
1160  REM       B(I)=COEFFICIENTS OF REGRESSION EQUATION
1170  REM       T95(I)=STATISTIC VALUES
1180  REM       R(I,J)=USED FOR ALL OF THE FOLLOWING
1190  REM           -SUM OF SQUARES AND CROSS PRODUCTS
1200  REM           -RESIDUAL SUMS OF SQUARES AND CROSS PRODUCTS
1210  REM           -SIMPLE CORRELATION COEFFICIENTS
1220  REM           -PARTIAL CORRELATION COEFFICIENTS
1230  REM           -AN INVERSE MATRIX
1240 DOS$ =  CHR$ (4): REM  CTRL-D
1250 NOBS = 30: REM  MAX.NO.OBSERVATIONS/VAR.
1260 NVAR = 11: REM  MAX. NO. VARIABLES
1270  DIM X(NVAR,NOBS),WT(NOBS),WSUM(NVAR),WMEAN(NVAR)
1280  DIM R(NVAR,NVAR),SIGMA(NVAR),B(NVAR),SB(NVAR)
1290  DIM VNAM$(NVAR),T95(33),Y(NOBS),E1(NVAR - 1,NVAR - 1)
1300  FOR I = 1 TO 33: REM  INITIALIZE T-STATISTIC ARRAY
1310  READ T95(I)
1320  NEXT I
1330  DATA  12.706, 4.303, 3.182, 2.776, 2.571, 2.447, 2.365
1340  DATA  2.306, 2.262, 2.228, 2.201, 2.179, 2.160, 2.145
1350  DATA  2.131,2.120, 2.110, 2.101, 2.093, 2.086, 2.080
1360  DATA  2.074, 2.069, 2.064, 2.060, 2.056, 2.052, 2.048
1370  DATA  2.045, 2.042, 2.021, 2.000, 1.980
1380  PRINT "****************************************"
1390  PRINT "MULTIPLE AND STEPWISE REGRESSION PROGRAM"
1400  PRINT "****************************************"
1410  FOR I = 1 TO NOBS: REM  ASSUME UNIFORM WEIGHTING
1420 WT(I) = 1
1430  NEXT I
1440  PRINT : INPUT "LIST PROGRAM OPTIONS (Y/N) ? ";Y$
1450  IF Y$ <  > "Y" THEN 1480
1460  GOSUB 7440
1470  GOTO 1480
1480  PRINT : PRINT
1490  INPUT "HOW ENTER DATA-KEYBOARD OR DISK (K/D)? ";K$
1500  IF (K$ <  > "K") AND (K$ <  > "D") THEN  PRINT "INVALID": GOTO 1490
1510  IF K$ = "K" THEN 1540
1520  GOSUB 6960
1530  GOTO 1910
1540  PRINT : INPUT "ENTER NAME OF DEPENDENT VARIABLE ? ";TEMP$
1550  PRINT : INPUT "ENTER NUMBER OF INDEPENDENT VARIABLES ?";IV
1560  IF (IV > 0) AND (IV < NVAR) THEN 1580
1570  PRINT : PRINT "MUST BE >0 AND <";NVAR: GOTO 1550
1580 VNAM$(IV + 1) = TEMP$
1590  FOR I = 1 TO IV
1600  PRINT : PRINT "ENTER NAME OF INDEPENDENT VARIABLE ";I;" :"
1610  INPUT VNAM$(I)
1620  NEXT I
1630  PRINT : INPUT "FORECAST REQUIRED (Y/N) ? ";F$
1640  IF (F$ <  > "Y") AND (F$ <  > "N") THEN  PRINT "INVALID": GOTO 1630
1650  IF F$ = "N" THEN I4 = 0: GOTO 1710
```

```
1660  PRINT : INPUT "NUMBER OF PERIODS TO BE FORECAST ? ";I4
1670  PRINT : PRINT "ENTER NUMBER OF DATA POINTS FOR EACH"
1680  PRINT "INDEPENDENT VARIABLE INCLUDING POINTS"
1690  INPUT "USED FOR FORECAST ? ";IO
1700  GOTO 1730
1710  PRINT : PRINT "ENTER NUMBER OF DATA POINTS FOR EACH"
1720  INPUT "INDEPENDENT VARIABLE ? ";IO
1730  IF IO >  = (IV + 2 + I4) THEN 1750
1740  PRINT : PRINT "NUMBER OF DATA POINTS MUST BE >";IV + 2 + I4: GOTO 1650
1750  IF IO <  = NOBS THEN 1770
1760  PRINT : PRINT "NUMBER OF DATA POINTS MUST BE < ";NOBS + 1: GOTO 1650
1770 IT = IV + 1:IH = IO - I4
1780  PRINT : PRINT "ENTER DATA"
1790  FOR I = 1 TO IT
1800  PRINT : IF I <  > IT THEN 1830
1810  PRINT IH;" VALUES REQUIRED FOR DEPENDENT VARIABLE ";VNAM$(IT)
1820  GOTO 1850
1830  PRINT  TAB( 7);IO;" VALUES REQUIRED FOR INDEPENDENT VARIABLE ";VNAM$(I): PRINT
1840 I5 = IO
1850  IF I = IT THEN I5 = IH
1860  FOR J = 1 TO I5
1870  PRINT "VALUE FOR POINT ";J;
1880  INPUT X(I,J)
1890  NEXT J
1900  NEXT I
1910  PRINT : PRINT  TAB( 5);"DATA MANAGEMENT OPTIONS:"
1920  PRINT  TAB( 5);"======================": PRINT
1930  PRINT  TAB( 10);"1-LIST DATA"
1940  PRINT  TAB( 10);"2-CORRECT DATA"
1950  PRINT  TAB( 10);"3-ADD TO DATA"
1960  PRINT  TAB( 10);"4-ADD AN INDEPENDENT VARIABLE"
1970  PRINT  TAB( 10);"5-DELETE A VARIABLE"
1980  PRINT  TAB( 10);"6-REGRESSION COMPUTATIONS"
1990  PRINT  TAB( 10);"7-STUDY ANOTHER MODEL"
2000  PRINT  TAB( 10);"8-STORE DATA ON DISK"
2010  PRINT  TAB( 10);"9-WEIGHT DATA"
2020  PRINT  TAB( 10);"10-TRANSFORM DATA"
2030  PRINT  TAB( 10);"11-QUIT"
2040  PRINT : INPUT "OPTION ? ";IP
2050  IF (IP < 1) OR (IP > 11) THEN  PRINT "INVALID": GOTO 1910
2060  IF IP = 1 THEN  GOSUB 5700
2070  IF IP = 2 THEN  GOSUB 5910
2080  IF IP = 3 THEN  GOSUB 6260
2090  IF IP = 4 THEN  GOSUB 6690
2100  IF IP = 5 THEN  GOSUB 6530
2110  IF IP = 6 THEN 2180
2120  IF IP = 7 THEN 1410
2130  IF IP = 8 THEN  GOSUB 6810
2140  IF IP = 9 THEN  GOSUB 7140
2150  IF IP = 10 THEN  GOSUB 7200
2160  IF IP = 11 THEN  PRINT : PRINT  TAB( 9);"*** END OF PROGRAM ***": END
2170  GOTO 1910
2180  PRINT : PRINT  TAB( 5);"AVAILABLE OPTIONS:"
2190  PRINT  TAB( 5);"------------------": PRINT
2200  PRINT  TAB( 8);"1- MULTIPLE REGRESSION"
2210  PRINT : PRINT  TAB( 8);"2- STEPWISE MULTIPLE REGRESSION"
2220  PRINT : PRINT  TAB( 8);"3- DATA MANAGEMENT OPTIONS": PRINT
2230  INPUT "OPTION ? ";IC
2240  IF (IC < 1) OR (IC > 3) THEN  PRINT "INVALID": GOTO 2180
2250  IF IC = 3 THEN 1910
2260 C$ = "Y"
2270  IF IC = 2 THEN C$ = "N"
2280 IR = 1
2290  PRINT : INPUT "PRINT SOLUTION AT EACH ITERATION (Y/N)? ";P$
2300  IF (P$ <  > "Y") AND (P$ <  > "N") THEN  PRINT "INVALID": GOTO 2290
2310  IF P$ = "N" THEN IR = IR + 1
```

```
2320  PRINT
2330  INPUT "OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? ";O$
2340  IF (O$ <  > "C") AND (O$ <  > "P") THEN  PRINT "INVALID": GOTO 2330
2350  PRINT : PRINT
2360  IF C$ = "Y" THEN 2400
2370  PRINT  TAB( 6);"STEPWISE MULTIPLE REGRESSION"
2380  PRINT  TAB( 5);"==========================="
2390  PRINT : PRINT : GOTO 2420
2400  PRINT  TAB( 10);"MULTIPLE REGRESSION"
2410  PRINT  TAB( 9);"====================": PRINT : PRINT
2420  IF O$ = "C" THEN 2500
2430  PRINT DOS$;"PR#1"
2440  IF C$ = "Y" THEN 2480
2450  PRINT  TAB( 10);"STEPWISE MULTIPLE REGRESSION"
2460  PRINT  TAB( 9);"==========================": PRINT
2470  GOTO 2500
2480  PRINT  TAB( 10);"MULTIPLE REGRESSION"
2490  PRINT  TAB( 9);"====================": PRINT
2500 F1 = 3.29: REM  F-TEST VALUE,VAR.ENTERING
2510 F2 = 3.29: REM   F-TEST VALUE,VAR.LEAVING
2520 TL = .0001
2530 IDS = 1
2540 ISTP = 1
2550 WDTA = 0
2560  FOR I = 1 TO IH
2570 WDTA = WDTA + WT(I): REM  TOTAL WEIGHTS
2580  NEXT I
2590  FOR I = 1 TO IT
2600 WSUM(I) = 0
2610  FOR J = 1 TO IH: REM   TOTAL WEIGHTED SUM
2620 WSUM(I) = WSUM(I) + WT(J) * X(I,J)
2630  NEXT J
2640  NEXT I
2650  REM  WEIGHTED SUMS OF SQUARES AND CROSS PRODUCTS
2660  FOR I = 1 TO IT
2670  FOR J = 1 TO IT
2680 R(I,J) = 0
2690  FOR K = 1 TO IH
2700 R(I,J) = R(I,J) + WT(K) * X(I,K) * X(J,K)
2710  NEXT K
2720  NEXT J
2730  NEXT I
2740  FOR I = 1 TO IT
2750 WMEAN(I) = WSUM(I) / WDTA: REM  WEIGHTED MEAN
2760  NEXT I
2770  REM  WEIGHTED RESIDUAL SUM OF SQUARES AND CROSS PRODUCTS
2780  FOR I = 1 TO IT
2790  FOR J = 1 TO IT
2800 R(I,J) = R(I,J) - WSUM(I) * WSUM(J) / WDTA
2810  NEXT J
2820  NEXT I
2830  FOR I = 1 TO IT
2840 SIGMA(I) = R(I,I) ^ .5
2850  NEXT I
2860  REM   CORRELATION COEFFICIENTS
2870  FOR I = 1 TO IT
2880  FOR J = 1 TO IT
2890 R(I,J) = R(I,J) / (SIGMA(I) * SIGMA(J))
2900  NEXT J
2910  NEXT I
2920  FOR I = 2 TO IT
2930 II = I - 1
2940  FOR J = 1 TO II
2950 R(I,J) = R(J,I)
2960  NEXT J
2970  NEXT I
```

```
2980  IF IC = 2 THEN 3050
2990  GOSUB 7840
3000  IF DET <  > 0 THEN 3050
3010  IF O$ = "P" THEN  PRINT DOS$;"PR#0"
3020  PRINT : PRINT "SOLUTION CANNOT BE FOUND USING MULTIPLE REGRESSION."
3030  PRINT "VARIABLES ARE NOT INDEPENDENT. TRY THE"
3040  PRINT "STEPWISE REGRESSION OPTION.": PRINT : GOTO 2180
3050 PHI = WDTA - 1
3060 INDEX = 1
3070  FOR J = 1 TO IT
3080 SB(J) = 0
3090 B(J) = 0
3100  NEXT J
3110  IF (R(IT,IT) >  - 9.999999E - 06) AND (R(IT,IT) < 0) THEN R(IT,IT) = 0
3120 SY = SIGMA(IT) * (R(IT,IT) / PHI) ^ .5: REM  STD.ERROR OF DEP.VAR.
3130  IF C$ <  > "Y" THEN 4010
3140  IF ISTP <  = 1 THEN 3300
3150 NN = ISTP - 1
3160  REM  REGRESSION COEFFICIENTS
3170  FOR I = 1 TO NN
3180 B(I) = R(I,IT) * SIGMA(IT) / SIGMA(I): REM  REGRESSION COEFFICIENTS
3190  IF R(I,I) > 0 THEN 3280
3200  PRINT DOS$;"PR#0"
3210  PRINT : PRINT " SOLUTION CAN NOT BE FOUND USING SIMPLE"
3220  PRINT "MULTIPLE REGRESSION. VARIABLE ";VNAM$(I);" IS A"
3230  PRINT "LINEAR COMBINATION OF THE OTHER"
3240  PRINT "VARIABLES."
3250  PRINT "TRY THE STEPWISE REGRESSION OPTION."
3260  PRINT : GOTO 2180
3270  REM  STD.ERROR OF REGRESSION COEFFICIENTS
3280 SB(I) = SY * R(I,I) ^ .5 / SIGMA(I)
3290  NEXT I
3300  IF IR > 1 THEN 3680: REM  SKIP PRINT
3310 B(IT) = WMEAN(IT)
3320 NN = IT - 1
3330  FOR I = 1 TO NN
3340 B(IT) = B(IT) - B(I) * WMEAN(I)
3350  NEXT I
3360  REM  COEFFICIENT OF DETERMINATION
3370 DETER = 1 - R(IT,IT)
3380  IF DETER < .000001 THEN DETER = 0
3390 DVEST = SY
3400  IF DVEST < .000001 THEN DVEST = 0
3410  IF O$ = "P" THEN 3590
3420  PRINT DOS$;"PR#0"
3430  PRINT "REGRESSION NUMBER ";ISTP
3440  PRINT "----------"
3450  PRINT  TAB( 5);VNAM$(IT);" = ";B(IT)
3460  FOR I = 1 TO NN
3470  IF B(I) = 0 THEN 3490
3480  PRINT " + ";B(I);" ";VNAM$(I)
3490  NEXT I
3500 DVEST =  INT (DVEST * 10 ^ 4 + .5) /  INT (10 ^ 4 + .5)
3510 DETER =  INT (DETER * 10 ^ 7 + .5) /  INT (10 ^ 7 + .5)
3520  PRINT "COEFFICIENT OF DETERMINATION = ";DETER
3530  PRINT "STD DEVIATION OF ESTIMATE = ";DVEST
3540  IF ISTP <  > (IDS * 3) THEN 3570
3550  IF ISTP = IT THEN 3680
3560 IDS = IDS + 1: PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
3570  PRINT : GOTO 3680
3580  PRINT DOS$;"PR#1"
3590  PRINT : PRINT "REGRESSION NUMBER ";ISTP
3600  PRINT  TAB( 5);VNAM$(IT);" = ";B(IT)
3610  FOR I = 1 TO NN
3620  IF B(I) = 0 THEN 3640
3630  PRINT " + ";B(I);" ";VNAM$(I)
```

```
3640  NEXT I
3650 DETER =  INT (DETER * 10 ^ 7 + .5) /  INT (10 ^ 7 + .5)
3660  PRINT "COEFFICIENT OF DETERMINATION = ";DETER
3670  PRINT "STD DEVIATION OF ESTIMATE = ";DVEST
3680 ISTP = ISTP + 1
3690  IF C$ <  > "Y" THEN 3840
3700  IF ISTP < IT THEN 3820
3710  IF ISTP > IT THEN 4370
3720  PRINT DOS$;"PR#0"
3730  IF O$ <  > "C" THEN  PRINT DOS$;"PR#1"
3740  PRINT : PRINT  TAB( 13);"FINAL SOLUTION": PRINT
3750  GOTO 3780
3760  PRINT DOS$;"PR#1"
3770  PRINT : PRINT "FINAL SOLUTION"
3780 IY = IR
3790 IR = 3
3800  IF C$ <  > "Y" THEN 3310
3810 IR = 1
3820 K = ISTP - 1
3830 PHI = PHI - 1
3840  IF IR > 2 THEN 4370
3850  REM  CALCULATE NEW MATRIX
3860  FOR I = 1 TO IT
3870  IF I = K THEN 3920
3880  FOR J = 1 TO IT
3890  IF J = K THEN 3910
3900 R(I,J) = R(I,J) - R(I,K) * R(K,J) / R(K,K)
3910  NEXT J
3920  NEXT I
3930  FOR I = 1 TO IT
3940  IF I = K THEN 3970
3950 R(I,K) =  - R(I,K) / R(K,K)
3960 R(K,I) = R(K,I) / R(K,K)
3970  NEXT I
3980 R(K,K) = 1 / R(K,K)
3990  IF C$ <  > "Y" THEN 3060
4000  GOTO 3110
4010 IX = 0
4020 VX = 0
4030 IM = 0
4040 VM = 99999
4050  IF (R(INDEX,INDEX) - TL) > 0 THEN 4200: REM  CHECK INDEPENDENCE
4060  IF (INDEX + 1) >  = IT THEN 4090
4070 INDEX = INDEX + 1
4080  GOTO 4050
4090  IF R(IT,IT) <  = 0 THEN 3720
4100  IF (VM * PHI / R(IT,IT)) >  = F2 THEN 4140: REM  VARIANCE SIGNIFICANT?
4110 K = IM
4120 PHI = PHI + 1
4130  GOTO 3300
4140  IF R(IT,IT) <  = VX THEN 4170
4150  REM  IS VARIANCE REDUCTION SIGNIFICANT?
4160  IF (VX * (PHI - 1) / (R(IT,IT) - VX)) <  = F1 THEN 3720
4170 K = IX
4180 PHI = PHI - 1
4190  GOTO 3300
4200 V = R(INDEX,IT) * R(IT,INDEX) / R(INDEX,INDEX)
4210  IF V = 0 THEN 4060: REM  NOT ADD TO EQUATION
4220  IF V > 0 THEN 4330: REM  MIGHT ADD TO EQUATION
4230  REM  REGRESSION COEFFICIENT FOR VARIABLE
4240 B(INDEX) = R(INDEX,IT) * SIGMA(IT) / SIGMA(INDEX)
4250  REM  VARIABLE STANDARD DEVIATION
4260 SB(INDEX) = SY * R(INDEX,INDEX) ^ .5 / SIGMA(INDEX)
4270  REM  WHAT VARIABLE CAUSES THE GREATEST VARIANCE REDUCTION,AND
4280  REM  WHAT VARIABLE CAUSES THE LEAST VARIANCE INCREASE?
4290  IF (V + VM) <  = 0 THEN 4060
```

```
4300 VM =  - V
4310 IM = INDEX
4320  GOTO 4060
4330  IF (V - VX) <  = 0 THEN 4060
4340 VX = V
4350 IX = INDEX
4360  GOTO 4060
4370  IF O$ = "P" THEN 4400
4380  PRINT DOS$;"PR#0"
4390  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
4400 IR = IY
4410 II = IT - 1
4420  REM  CALCULATE ESTIMATED VALUES FOR DEPENDENT VARIABLE
4430  FOR J = 1 TO IH
4440 Y(J) = B(IT)
4450  FOR I = 1 TO II
4460 Y(J) = Y(J) + B(I) * X(I,J)
4470  NEXT I
4480  NEXT J
4490 IGI = II - 1
4500  PRINT DOS$;"PR#0"
4510  PRINT : INPUT "WANT TO SEE LIMITS ON PREDICTION (Y/N)?";F$
4520  IF F$ <  > "Y" THEN 4740
4530  PRINT "USE PROGRAM SUPPLIED T-STATISTICS"
4540  INPUT "FOR LIMITS (Y/N) ? ";L$
4550  IF (L$ <  > "Y") AND (L$ <  > "N") THEN  PRINT "INVALID": GOTO 4530
4560  IF L$ = "Y" THEN 4600
4570  INPUT "ENTER THE T-STATISTIC ? ";T
4580  GOTO 4740
4590  REM  IF T-STATISTIC NOT IN ARRAY T95,THEN CALCULATE VALUE
4600  IF PHI > 30 THEN 4640
4610 IBI = PHI
4620 T = T95(IBI)
4630  GOTO 4740
4640  IF PHI > 40 THEN 4670
4650 T = T95(30) - (PHI - 30) * .0021
4660  GOTO 4740
4670  IF PHI > 60 THEN 4700
4680 T = T95(31) - (PHI - 40) * .00105
4690  GOTO 4740
4700  IF PHI > 120 THEN 4730
4710 T = T95(32) - (PHI - 60) * .02 / 60
4720  GOTO 4740
4730 T = T95(33)
4740 SAS = SIGMA(IT) ^ 2 * R(IT,IT) / PHI
4750  IF O$ = "P" THEN 5020
4760  PRINT : PRINT "ACTUAL VERSUS PREDICTED VALUES FOR ";VNAM$(IT)
4770  PRINT "ACTUAL";"     ";"PREDICTED";"     ";"DIFFERENCE"
4780  PRINT "-----------------------------------"
4790 IC = 1
4800  FOR J = 1 TO IH
4810  IF J <  > (IC * 20) THEN 4840
4820 IC = IC + 1
4830  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
4840 DI = X(IT,J) - Y(J)
4850  IF X(IT,J) <  > 0 THEN 4880
4860  IF DI <  > 0 THEN PDI = 999999
4870  GOTO 4890
4880 PDI = DI / X(IT,J) * 100
4890 OBS = X(IT,J)
4900 Y(J) =  INT (1000 * Y(J) + .5) / 1000
4910 DI =  INT (10 ^ 5 * DI + .5) / 10 ^ 5
4920  PRINT  TAB( 2);OBS; TAB( 12);Y(J); TAB( 26);DI
4930  NEXT J
4940  IF F$ <  > "Y" THEN 5680
4950  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
```

```
4960  PRINT : PRINT "CONFIDENCE LIMITS ON PREDICTED VALUES"
4970  PRINT : PRINT "DEGREES OF FREEDOM = ";PHI
4980  PRINT : PRINT "T-STATISTIC = ";T
4990  PRINT : PRINT "LOWER LIMIT    PREDICTED    UPPER LIMIT"
5000  PRINT "----------------------------------------"
5010  GOTO 5190
5020  PRINT DOS$;"PR#1"
5030  PRINT "ACTUAL","PREDICTED","DIFFERENCE","%DIFFERENCE"
5040  FOR J = 1 TO IH
5050 DI = X(IT,J) - Y(J)
5060  IF X(IT,J) <  > 0 THEN 5090
5070  IF DI <  > 0 THEN PDI = 999999
5080  GOTO 5110
5090 OBS = X(IT,J)
5100 PDI = DI / X(IT,J) * 100
5110  PRINT OBS,Y(J),DI,PDI
5120  NEXT J
5130  IF F$ <  > "Y" THEN 5680
5140  PRINT : PRINT "CONFIDENCE LIMITS ON PREDICTED VALUES"
5150  PRINT : PRINT "DEGREES OF FREEDOM = ";PHI
5160  PRINT : PRINT "T-STATISTIC = ";T
5170  PRINT : PRINT "LOWER LIMIT    PREDICTED    UPPER LIMIT"
5180  PRINT "----------------------------------------"
5190 XM = IH
5200 IC = 1
5210  FOR I = 1 TO IO
5220  IF I <  > (IC * 20) THEN 5270
5230 IC = IC + 1
5240  IF O$ <  > "C" THEN 5270
5250  PRINT DOS$;"PR#0"
5260  PRINT : INPUT #PRESS RETURN  TO  CONT INUE -  > ";Y$:PRINT
5270 VARY = SAS / XM
5280 Y(I) = B(IT)
5290  IF IGI <  = 0 THEN 5500
5300  FOR J = 1 TO IGI
5310  IF SB(J) = 0 THEN 5350
5320  IF (SIGMA(J) = 0) OR (SIGMA(J) > 1E + 20) THEN 5350
5330 XSB = SAS * R(J,J) / (SIGMA(J) ^ 2)
5340  GOTO 5360
5350 XSB = 0
5360 Y(I) = Y(I) + B(J) * X(J,I)
5370 DI = X(J,I) - WMEAN(J)
5380 VARY = VARY + XSB * DI ^ 2
5390 KK = J + 1
5400  FOR K = KK TO II
5410  IF SB(K) = 0 THEN 5460
5420  IF (SIGMA(J) = 0) OR (SIGMA(J) >  = 1E + 20) THEN 5460
5430  IF (SIGMA(K) = 0) OR (SIGMA(K) >  = 1E + 20) THEN 5460
5440 XSB = SAS * R(J,K) / (SIGMA(J) * SIGMA(K))
5450  GOTO 5470
5460 XSB = 0
5470 VARY = VARY + 2 * XSB * DI * (X(K,I) - WMEAN(K))
5480  NEXT K
5490  NEXT J
5500  IF SB(II) <  = 0 THEN 5540
5510  IF (SIGMA(II) = 0) OR (SIGMA(II) >  = 1E + 20) THEN 5540
5520 XSB = SAS * R(II,II) / SIGMA(II) ^ 2
5530  GOTO 5550
5540 XSB = 0
5550 Y(I) = Y(I) + B(II) * X(II,I)
5560 VARY = VARY + XSB * (X(II,I) - WMEAN(II)) ^ 2
5570 SVARY = (VARY + SAS) ^ .5
5580 LLM = Y(I) - SVARY * T
5590 ULM = Y(I) + SVARY * T
5600  IF O$ = "P" THEN 5630
5610  PRINT  TAB( 2);LLM; TAB( 16);Y(I); TAB( 29);ULM
```

```
5620  GOTO 5650
5630  PRINT DOS$;"PR#1"
5640  PRINT LLM,Y(I),ULM
5650  NEXT I
5660  PRINT DOS$;"PR#0"
5670  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
5680  IF O$ = "P" THEN  PRINT DOS$;"PR#0"
5690  GOTO 2180
5700  REM  SUBROUTINE: LIST DATA
5710  PRINT : INPUT "ENTER NAME OF VARIABLE IDENTIFYING DATA TO BE LISTED ? ";ID$
5720  GOSUB 6190
5730  IF IL = 0 THEN 5770
5740  PRINT : INPUT "NO MATCH - TRY AGAIN (Y/N) ? ";Y$
5750  IF Y$ <  > "Y" THEN  RETURN
5760  GOTO 5710
5770 MM = IO
5780  IF I = IT THEN MM = IH
5790  PRINT : PRINT "LIST OF DATA FOR ";VNAM$(I);" :"
5800  PRINT "----------------------": PRINT
5810 IC = 1
5820  FOR J = 1 TO MM
5830  IF IC <  > (IC * 20) THEN 5860
5840 IC = IC + 1
5850  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
5860  PRINT X(I,J)
5870  NEXT J
5880  PRINT : INPUT "LIST ADDITIONAL DATA (Y/N) ? ";Y$
5890  IF Y$ = "Y" THEN 5700
5900  RETURN
5910  REM  SUBROUTINE: CORRECT DATA
5920  PRINT : PRINT "ENTER VARIABLE NAME SPECIFYING DATA TO"
5930  INPUT "BE CHANGED ? ";ID$
5940  GOSUB 6190
5950  IF IL = 0 THEN 5990
5960  PRINT : INPUT "NO MATCH - TRY AGAIN (Y/N) ? ";Y$
5970  IF Y$ <  > "Y" THEN  RETURN
5980  GOTO 5920
5990 I5 = IO
6000  IF I = IT THEN I5 = IH
6010  PRINT : INPUT "DO YOU WANT TO MODIFY ALL DATA FOR THIS VARIABLE (Y/N) ? ";Y$
6020  IF Y$ = "Y" THEN 6110
6030  PRINT : PRINT "ENTER POSITION OF DATA ELEMENT TO BE"
6040  INPUT "CHANGED ? ";J
6050  IF (J < 1) OR (J > I5) THEN  PRINT "INVALID NO.": GOTO 6030
6060  PRINT : INPUT "ENTER DATA ELEMENT ? ";X(I,J)
6070  PRINT : PRINT "WANT TO CHANGE OTHER DATA FOR THIS"
6080  INPUT "VARIABLE (Y/N) ? ";Y$
6090  IF Y$ = "Y" THEN 6030
6100  RETURN
6110 I5 = IO
6120  IF I = IT THEN I5 = IH
6130  PRINT : PRINT I5;" VALUES REQUIRED FOR ";VNAM$(I): PRINT
6140  FOR J = 1 TO I5
6150  PRINT " ENTER VALUE FOR POINT ";J;
6160  INPUT X(I,J)
6170  NEXT J
6180  GOTO 6070
6190  REM  SUBROUTINE: LOCATE MATCHING VARIABLE NAME
6200 IL = 0
6210  FOR I = 1 TO IT
6220  IF ID$ = VNAM$(I) THEN  RETURN
6230  NEXT I
6240 IL = 1: REM  NO MATCH FOUND
6250  RETURN
6260  REM  SUBROUTINE: ADD TO DATA
6270  PRINT : INPUT "FORECAST REQUIRED (Y/N) ? ";F$: PRINT
```

```
6280  IF (F$ < > "Y") AND (F$ < > "N") THEN  PRINT "INVALID": GOTO 6270
6290  IF F$ = "N" THEN I4 = 0: GOTO 6310
6300  INPUT "NUMBER OF PERIODS TO BE FORECAST ? ";I4: PRINT
6310  PRINT "ENTER NUMBER DATA POINTS FOR EACH"
6320  PRINT "INDEPENDENT VARIABLE INCLUDING POINTS"
6330  INPUT "USED FOR FORECAST ? ";IA
6340  IF IA > (IV + 2 - I4) THEN 6360
6350  PRINT : PRINT "NUMBER DATA POINTS MUST BE >";IV + 2 - I4: GOTO 6310
6360  IF IA <  = NOBS THEN I1 = IH + 1:I2 = IA: GOTO 6380
6370  PRINT : PRINT "NUMBER DATA POINTS MUST BE <";NOBS + 1: GOTO 6310
6380  FOR I = 1 TO IT
6390  IF I = IT THEN I1 = IH + 1:I2 = IA - I4
6400  IF I < > IT THEN 6440
6410  PRINT : PRINT "ENTER ADDITIONAL DATA FOR DEPENDENT"
6420  PRINT "VARIABLE ";VNAM$(IT)
6430  GOTO 6460
6440  PRINT : PRINT "ENTER ADDITIONAL DATA FOR INDEPENDENT"
6450  PRINT "VARIABLE ";VNAM$(I)
6460  FOR J = I1 TO I2
6470  PRINT "VALUE FOR POINT ";J;
6480  INPUT X(I,J)
6490  NEXT J
6500  NEXT I
6510 IO = IA:IH = IO - I4
6520  RETURN
6530  REM  SUBROUTINE: DELETE AN INDEPENDENT VARIABLE
6540  PRINT : INPUT "ENTER NAME OF VARIABLE TO BE DELETED ? ";ID$: PRINT
6550  GOSUB 6190
6560  IF IL = 0 THEN 6600
6570  INPUT "NO MATCH - TRY AGAIN (Y/N) ? ";Y$
6580  IF Y$ < > "Y" THEN  RETURN
6590  GOTO 6540
6600  FOR J = I TO IT - 1
6610 VNAM$(J) = VNAM$(J + 1)
6620  FOR K = 1 TO IO
6630 X(J,K) = X(J + 1,K)
6640  NEXT K
6650  NEXT J
6660 IV = IV - 1
6670 IT = IT - 1
6680  RETURN
6690  REM  SUBROUTINE: ADD AN INDEPENDENT VARIABLE
6700 VNAM$(IT + 1) = VNAM$(IT)
6710  PRINT : INPUT "ENTER NAME OF VARIABLE TO BE ADDED ? ";VNAM$(IT): PRINT
6720  PRINT IO;" VALUES REQUIRED FOR ";VNAM$(IT)
6730  FOR J = 1 TO IO
6740 X(IT + 1,J) = X(IT,J)
6750  PRINT " VALUE FOR POINT ";J;
6760  INPUT X(IT,J)
6770  NEXT J
6780 IT = IT + 1
6790 IV = IV + 1
6800  RETURN
6810  REM  SUBROUTINE: STORE DATA ON DISK
6820  PRINT : INPUT "ENTER NAME OF DISK: FILE ? ";NAM$
6830  PRINT DOS$;"OPEN";NAM$
6840  PRINT DOS$;"WRITE";NAM$
6850  PRINT IT: PRINT IH: PRINT I4
6860 ID = IO
6870  FOR I = 1 TO IT
6880  PRINT VNAM$(I)
6890  IF I = IT THEN ID = IH
6900  FOR J = 1 TO ID
6910  PRINT X(I,J)
6920  NEXT J
6930  NEXT I
```

```
6940  PRINT DOS$;"CLOSE";NAM$
6950  RETURN
6960  REM  SUBROUTINE: READ DATA FROM DISK
6970  INPUT "ENTER NAME OF DISK:FILE ? ";NAM$
6980  PRINT DOS$;"OPEN";NAM$
6990  PRINT DOS$;"READ";NAM$
7000  INPUT IT: INPUT IH: INPUT I4
7010 IO = IH + I4
7020 IV = IT - 1
7030  IF I4 > 0 THEN F$ = "Y"
7040 ID = IO
7050  FOR I = 1 TO IT
7060  INPUT VNAM$(I)
7070  IF (I = IT) THEN ID = IH
7080  FOR J = 1 TO ID
7090  INPUT X(I,J)
7100  NEXT J
7110  NEXT I
7120  PRINT DOS$;"CLOSE";NAM$
7130  RETURN
7140  REM  SUBROUTINE: WEIGHT DATA
7150  FOR I = 1 TO IH
7160  PRINT "ENTER WEIGHT FOR DATA POINT ";I;
7170  INPUT WT(I)
7180  NEXT I
7190  RETURN
7200  REM  SUBROUTINE: DATA TRANSFORM
7210  GOSUB 7790
7220 MM = IO
7230  FOR I = 1 TO IT
7240  IF I = IT THEN MM = IH
7250  PRINT : PRINT "ENTER TRANSFORM CODE FOR VARIABLE ";VNAM$(I);
7260  INPUT TC$
7270  IF TC$ = "N" THEN 7420
7280  IF TC$ = "E" THEN IK = 1: GOTO 7320
7290  IF TC$ = "P" THEN IK = 2: GOTO 7310
7300  PRINT "INVALID CODE": GOTO 7250
7310  PRINT : INPUT "ENTER VALUE FOR EXPONENT (X^N) ? ";EX: PRINT
7320  FOR J = 1 TO MM: REM  PERFORM TRANSFORMS
7330  ON IK GOTO 7340,7400
7340  IF X(I,J) > 0 THEN 7380
7350  PRINT "CANNOT TAKE THE LOG OF A NUMBER <=0 "
7360  PRINT "VALUE OF DATA ELEMENT INVOLVED IS ";X(I,J)
7370  PRINT : GOTO 7410
7380 X(I,J) =  LOG (X(I,J))
7390  GOTO 7410
7400 X(I,J) = X(I,J) ^ EX
7410  NEXT J
7420  NEXT I
7430  RETURN
7440  REM  SUBROUTINE: LIST PROGRAM OPTIONS
7450  PRINT : PRINT "PROGRAM OPTIONS AVAILABLE:"
7460  PRINT "==========================="
7470  PRINT  TAB( 3);"DATA INPUT:"
7480  PRINT : PRINT  TAB( 5);"K = KEYBOARD"
7490  PRINT  TAB( 5);"D = DISK FILE"
7500  PRINT : PRINT  TAB( 3);"DATA MANAGEMENT:"
7510  PRINT : PRINT  TAB( 5);" CORRECTION OF DATA"
7520  PRINT  TAB( 5);" ADD TO DATA"
7530  PRINT  TAB( 5);" DELETION OF VARIABLES"
7540  PRINT  TAB( 5);" ADDITION OF VARIABLES"
7550  PRINT  TAB( 5);" WEIGHT DATA"
7560  PRINT  TAB( 5);" TRANSFORM DATA"
```

```
7570  PRINT  TAB( 5);" STORE DATA ON DISK"
7580  PRINT : PRINT  TAB( 3);"OUTPUT RESULTS:"
7590  PRINT : PRINT  TAB( 5);"C = CRT DISPLAY"
7600  PRINT  TAB( 5);"P = PRINTER"
7610  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
7620  PRINT  TAB( 34);"CONT.": PRINT
7630  PRINT : PRINT  TAB( 3);"COMPUTATIONAL:"
7640  PRINT : PRINT  TAB( 5);" MULTIPLE REGRESSION"
7650  PRINT  TAB( 5);" STEPWISE MULTIPLE REGRESSION"
7660  PRINT : PRINT  TAB( 3);"SOLUTION RESULTS:"
7670  PRINT : PRINT  TAB( 5);"Y = SOLUTION AT EACH ITERATION"
7680  PRINT  TAB( 5);"N = FINAL SOLUTION ONLY"
7690  PRINT : PRINT  TAB( 3);"DATA WEIGHTING SCHEMES:"
7700  PRINT : PRINT  TAB( 5);" UNIFORM,PROGRAM SUPPLIED"
7710  PRINT  TAB( 5);" NON-UNIFORM,USER SUPPLIED WEIGHTS"
7720  PRINT : PRINT  TAB( 3);"CONFIDENCE LIMITS:"
7730  PRINT : PRINT  TAB( 5);"Y=PROGRAM SUPPLIED 95% T-STATISTIC"
7740  PRINT  TAB( 5);"N = USER SUPPLIED T-STATISTIC"
7750  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
7760  PRINT : PRINT  TAB( 3);"FORECAST HORIZON:"
7770  PRINT : PRINT  TAB( 5);"Y = FORECAST REQUIRED"
7780  PRINT  TAB( 5);"N = NO FORECAST"
7790  PRINT : PRINT  TAB( 3);"TRANSFORMS AVAILABLE:"
7800  PRINT : PRINT  TAB( 5);"N = NO TRANSFORMS"
7810  PRINT  TAB( 5);"E = LOG(X) TO BASE E"
7820  PRINT  TAB( 5);"P = POLYNOMIAL X^N"
7830  RETURN
7840  REM  SUBROUTINE: CALCULATE DETERMINANT
7850  FOR I = 1 TO IV
7860  FOR J = 1 TO IV
7870  REM  ASSIGN R MATRIX TO E1
7880 E1(I,J) = R(I,J)
7890  NEXT J
7900  NEXT I
7910  REM  SET DETERMINANT FLAG
7920 DET = 1
7930  FOR I = 1 TO IV - 1
7940  REM  DIAGONAL ELEMENT MUST NOT BE ZERO
7950  IF E1(I,I) = 0 THEN  GOSUB 8110
7960  IF DET = 0 THEN  RETURN
7970  FOR J = I + 1 TO IV
7980  REM  MULTIPLIER TO ZERO THE COLUMN ELEMENTS
7990 XM = E1(J,I) / E1(I,I)
8000  FOR K = I TO IV
8010  REM  CALCULATE NEW ELEMENTS
8020 E1(J,K) = E1(J,K) - XM * E1(I,K)
8030  NEXT K
8040  NEXT J
8050  NEXT I
8060  REM  CALCULATE DETERMINANT
8070  FOR I = 1 TO IV
8080 DET = DET * E1(I,I)
8090  NEXT I
8100  RETURN
8110  REM  SUBROUTINE: ZERO DETERMINANT DETECTION
8120  REM  SUBROUTINE TO ENSURE DIAGONAL ELEMENTS ARE NONZERO DURING
8130  REM  UPPER-TRIANGULARIZATION. IF THIS IS NOT POSSIBLE, DET IS ZERO.
8140  FOR J = J + 1 TO IV
8150  IF E1(J,I) = 0 THEN  GOTO 8210
8160  FOR K = 1 TO IV
8170  REM  ADD ROWS TO MAKE DIAGONAL NONZERO
8180 E1(I,K) = E1(I,K) + E1(J,K)
```

```
8190  NEXT K
8200  RETURN
8210  NEXT J
8220  REM  DET MUST BE ZERO AT THIS POINT
8230 DET = 0
8240  RETURN
```

Figure 4.4—Stepwise Linear Regression Program

references Draper, (1966) and Ralston, (1964). However, we will explain what the program can do and present some application examples.

Program Options

As execution of the program begins, the user is given an opportunity to receive a listing of the program options. Figure 4.5 contains a list of these options as they would appear on your display. A brief explanation of these options will be given and then each one will be illustrated using examples. Data can be input in two ways, from the keyboard or from a previously saved disk file. As soon as the program options are listed, the program prompts for how data are to be entered. A letter appearing to the right of an option, such as

K = KEYBOARD,

means that a user response of K at the appropriate time will denote that data are to be entered from the keyboard. Options appearing in Figure 4.5 without a letter in front are selected by a number from a list of options displayed on the screen at various times.

```
LOAD STEPWISE
RUN

**************************************
MULTIPLE AND STEPWISE REGRESSION PROGRAM
**************************************

LIST PROGRAM OPTIONS (Y/N) ? Y

PROGRAM OPTIONS AVAILABLE:
 ========================

  DATA INPUT:

      K = KEYBOARD
      D = DISK FILE

  DATA MANAGEMENT:
      CORRECTION OF DATA
      ADD TO DATA
      DELETION OF VARIABLES
      ADDITION OF VARIABLES
      WEIGHT DATA
      TRANSFORM DATA
      STORE DATA ON DISK
```

```
OUTPUT RESULTS:

    C=CRT DISPLAY
    P=PRINTER

COMPUTATIONAL:

    MULTIPLE REGRESSION
    STEPWISE MULTIPLE REGRESSION

SOLUTION RESULTS:

    Y = SOLUTION AT EACH ITERATION
    N = FINAL SOLUTION ONLY

DATA WEIGHTING SCHEMES:

    UNIFORM, PROGRAM SUPPLIED
    NON-UNIFORM, USER SUPPLIED WEIGHTS

CONFIDENCE LIMITS:

    Y = PROGRAM SUPPLIED 95% T-STATISTIC
    N = USER SUPPLIED T-STATISTIC

FORECAST HORIZON:

    Y = FORECAST REQUIRED
    N = NO FORECAST

TRANSFORMS AVAILABLE:

    N = NO TRANSFORMS
    E = LOG(X) TO BASE E
    P = POLYNOMIAL X^N
```

Figure 4.5—Program Options

Several data management options are provided. After data are input, the user is given an opportunity to select any of these data management options. Data can be listed for any variable by specifying the variable name. Using the option CORRECT DATA, any data element or all data elements for a particular variable can be changed. Additional data elements can be added to a set of data by using the ADD TO DATA option. Thus, you might initially study a problem using nine data elements for each variable. However, in an attempt to improve the regression equation fit, you may append one or more data elements to the set of observations associated with each variable.

Any independent variable and its associated data can be removed from the problem being studied using the DELETE INDEPENDENT VARIABLE option. All data are stored in the two dimensional array X(I,J). Data for the independent variables are stored first, one variable per row. Data for the dependent variable are stored in row N + 1, where N is the number of independent variables. Consequently, if variable N is deleted from the problem, data for the dependent variable are moved from row N+1 to row N. Similarly, an independent variable can be added to the problem using option ADD INDEPENDENT VARIABLE. Also, if you want to keep data for further analysis, you may store it in a disk file to avoid having to reenter the data from the keyboard.

Output from the regression analysis can be displayed on the CRT or at the printer. If output is displayed at the CRT, program execution is periodically stopped to give the user an opportunity to read the information being displayed before it is scrolled off the display. To continue program execution, the user is prompted to press the "RETURN" key.

Two regression computation options are provided: multiple regression and stepwise regression. Multiple regression means that all independent variables will appear in the final regression equation. However, only one variable is added at a time to the regression equation. If the user specifies that the solution is to be printed at each iteration, then the regression equation, the coefficient of determination, and the standard deviation of the estimate will be printed as each new variable is added to the regression equation. If stepwise multiple regression is selected, only the independent variables that significantly reduce the variance will appear in the regression equation.

Usually, each data element in a set of data is given equal weight. However, there may be instances when some of the elements may be more accurate, such as the more recent elements in a time series. The weighting option lets the user specify weights for each data element. It is best that the weights be normalized such that the average weight is one.

Included in the final solution are confidence limits on the predicted values. These limits are calculated using a 95% t-statistic. Given a predicted value for the dependent variable, we should expect that 95% of the time the actual value will fall within these predicted limits. If other limits are desired, the user can specify the t-statistic values that are to be employed.

As noted earlier in this chapter, some nonlinear expressions can be transformed so that linear regression can be applied. If specified, the program can perform the following data transformations:

1) $A\ e^{Bt}$ to $\ln A + Bt$
2) X^2 to X'

Once the user specifies that transformations are required, the program will prompt for the needed input. If other transformations are required (in addition to the two provided by the program), the user will have to transform the data before they are input. As each data element is transformed, it replaces the original input data; therefore, once data are transformed, they should not be transformed again during a subsequent run through the regression program.

Forecasting is a common application of linear regression. To facilitate such usage, the user can specify the number of values to be forecast. For each period to be forecast, an observed value must be entered for each independent variable. Several examples will be presented to illustrate how to use the options of the multiple and stepwise regression program. Output from the program will be presented as it appears on the monitor or printer. User responses are denoted by underlined characters. Output from the computer is denoted by characters without the underline.

Using the Stepwise Program

EXAMPLE 2. For the past six months, the following data had been compiled at the Brady Company. Using the data from Example 1 and the multiple and stepwise regression program, forecast the sales for period seven.

Month	Orders on Hand at Beginning of Month	Sales for the Month
1	$3550	$8350
2	3600	8730
3	3250	8120
4	3350	8450
5	3000	7900
6	3590	8650

At the beginning of the seventh month, the orders on hand had a value of $3700. What would your sales forecast be for the seventh month?

SOLUTION. Because the multiple and stepwise regression program provides several options, the entire output from running the program will be presented here. We will intersperse some comments within this output to assist in explaining how to use this program.

The multiple and stepwise program is stored in the file STEPWISE; therefore, we start by loading this file.

```
LOAD STEPWISE
RUN

**************************************
MULTIPLE AND STEPWISE REGRESSION PROGRAM
**************************************

LIST PROGRAM OPTIONS (Y/N) ? Y

PROGRAM OPTIONS AVAILABLE:
 ========================
  DATA INPUT:

      K = KEYBOARD
      D = DISK FILE

  DATA MANAGEMENT:
      CORRECTION OF DATA
      ADD TO DATA
      DELETION OF VARIABLES
      ADDITION OF VARIABLES
      WEIGHT DATA
      TRANSFORM DATA
      STORE DATA ON DISK


  OUTPUT RESULTS:

      C=CRT DISPLAY
      P=PRINTER

PRESS RETURN TO CONTINUE ->
```

```
  COMPUTATIONAL:

     MULTIPLE REGRESSION
     STEPWISE MULTIPLE REGRESSION

  SOLUTION RESULTS:

     Y = SOLUTION AT EACH ITERATION
     N = FINAL SOLUTION ONLY

  DATA WEIGHTING SCHEMES:

     UNIFORM, PROGRAM SUPPLIED
     NON-UNIFORM, USER SUPPLIED WEIGHTS

  CONFIDENCE LIMITS:

     Y = PROGRAM SUPPLIED 95% T-STATISTIC
     N = USER SUPPLIED T-STATISTIC

PRESS RETURN TO CONTINUE ->

  FORECAST HORIZON:

     Y = FORECAST REQUIRED
     N = NO FORECAST

  TRANSFORMS AVAILABLE:

     N = NO TRANSFORMS
     E = LOG(X) TO BASE E
     P = POLYNOMIAL X^N
```

This list of options describes the capabilities of the program. Note the "PRESS RETURN TO CONTINUE" phase; this stops program execution so that the user can read the displayed information. Also note that some of the options have a character appearing on the left. These characters are possible responses to a prompting question, and the narrative to the right of the character describes what will happen when the designated character is entered.

Proceeding with this solution, we are asked to specify how we want to enter our data. We enter a K denoting that the data will be entered from the keyboard. We also specify that the name of our dependent variables is SALES; we have 1 independent variable named ORDERS; we want to forecast for 1 period, and we have 7 data points for the independent variable. After this information is input, the program prompts for the required data.

```
HOW ENTER DATA-KEYBOARD OR DISK (K/D)? K

ENTER NAME OF DEPENDENT VARIABLE ? SALES

ENTER NUMBER OF INDEPENDENT VARIABLES: 1

ENTER NAME OF INDEPENDENT VARIABLE 1 :
?ORDERS
```

```
FORECAST REQUIRED (Y/N) ? Y

NUMBER OF PERIODS TO BE FORECAST ? 1

ENTER NUMBER OF DATA POINTS FOR EACH
INDEPENDENT VARIABLE INCLUDING POINTS
USED FOR FORECAST ? 7

ENTER DATA

   7 VALUES REQUIRED FOR INDEPENDENT
VARIABLE ORDERS

VALUE FOR POINT 1?3550
VALUE FOR POINT 2?360
VALUE FOR POINT 3?3250
VALUE FOR POINT 4?3350
VALUE FOR POINT 5?3000
VALUE FOR POINT 6?3590
VALUE FOR POINT 7?3700

6 VALUES REQUIRED FOR DEPENDENT  VARIABLE
SALES
VALUE FOR POINT 1?8350
VALUE FOR POINT 2?8730
VALUE FOR POINT 3?8120
VALUE FOR POINT 4?8450
VALUE FOR POINT 5?7900
VALUE FOR POINT 6?8650
```

The independent variable requires one more data element than is required by the dependent variable. The first six data elements of the dependent and independent variables are used to develop the regression equation. The seventh independent variable data element is used to forecast the seventh data element for the dependent variable (sales for period seven).

After the data are input, the program provides a list of data management options. We specify option 1, LIST DATA, because we think that one of the data elements for ORDERS was input incorrectly. This option proceeds to ask for the name of the variable for which we want to list data. We respond with the name ORDERS.

```
DATA MANAGEMENT OPTIONS:
========================

          1-LIST DATA
          2-CORRECT DATA
          3-ADD TO DATA
          4-ADD AN INDEPENDENT VARIABLE
          5-DELETE A VARIABLE
          6-REGRESSION COMPUTATIONS
          7-STUDY ANOTHER MODEL
          8-STORE DATA ON DISK
          9-WEIGHT DATA
         10-TRANSFORM DATA
         11-QUIT
```

```
OPTION ? 1

ENTER NAME OF VARIABLE IDENTIFYING DATA
TO BE LISTED ? ORDERS

LIST OF DATA FOR ORDERS:

3550
360
3250
3350
3000
3590
3700
```

Looking at the data, we can see that element two is wrong; it should be 3600. Therefore, we need to use option 2, CORRECT DATA.

```
LIST ADDITIONAL DATA (Y/N) ? N

      DATA MANAGEMENT OPTIONS:
      ========================

        1-LIST DATA
        2-CORRECT DATA
        3-ADD TO DATA
        4-ADD AN INDEPENDENT VARIABLE
        5-DELETE A VARIABLE
        6-PERFORM REGRESSION COMPUTATIONS
        7-STUDY ANOTHER MODEL
        8-STORE DATA ON DISK
        9-WEIGHT DATA
       10-TRANSFORM DATA
       11-QUIT

OPTION ? 2

ENTER VARIABLE NAME SPECIFYING DATA TO
BE CHANGED ? ORDERS

DO YOU WANT TO MODIFY ALL DATA FOR THIS
VARIABLE (Y/N) ? N

ENTER POSITION OF DATA ELEMENT TO BE
CHANGED ? 2

ENTER DATA ELEMENT ? 3600
```

Note that it is possible to change all of the data elements associated with a variable. However, in this case, we only need to change element two. Having changed the known incorrect data, we will list the data associated with each variable to verify that every element is correct.

```
WANT TO CHANGE OTHER DATA FOR THIS VARIABLE (Y/N) ? N
```

```
      DATA MANAGEMENT OPTIONS:
      ========================

        1-LIST DATA
        2-CORRECT DATA
        3-ADD TO DATA
        4-ADD AN INDEPENDENT VARIABLE
        5-DELETE A VARIABLE
        6-PERFORM REGRESSION COMPUTATIONS
        7-STUDY ANOTHER MODEL
        8-STORE DATA ON DISK
        9-WEIGHT DATA
       10-TRANSFORM DATA
       11-QUIT

OPTION ? 1

ENTER VARIABLE NAME SPECIFYING DATA TO
BE LISTED ? ORDERS:

LIST OF DATA FOR ORDERS:

 3550
 3600
 3250
 3350
 3000
 3590
 3700

LIST ADDITIONAL DATA (Y/N)? Y
ENTER NAME OF VARIABLE IDENTIFYING DATA
TO BE LISTED ? SALES

LIST OF DATA FOR SALES:
 8350
 8730
 8120
 8450
 7900
 8650
```

All of the data are correct. Before proceeding with the regression analysis, we will store these data in a disk file. By doing this, we will avoid having to reenter these data from the keyboard if we want to use these data at a later time. Using option 8, STORE DATA ON DISK, we store the data for ORDERS and SALES in a file called BRASALES.

```
LIST ADDITIONAL DATA (Y/N) ? N

      DATA MANAGEMENT OPTIONS:
      ========================

        1-LIST DATA
        2-CORRECT DATA
        3-ADD TO DATA
        4-ADD AN INDEPENDENT VARIABLE
        5-DELETE A VARIABLE
        6-REGRESSION COMPUTATIONS
```

```
 7-STUDY ANOTHER MODEL
 8-STORE DATA ON DISK
 9-WEIGHT DATA
10-TRANSFORM DATA
11-QUIT

OPTION ? 8

ENTER NAME OF DISK: FILE ? BRASALES

      DATA MANAGEMENT OPTIONS:
      ========================

        1-LIST DATA
        2-CORRECT DATA
        3-ADD TO DATA
        4-ADD AN INDEPENDENT VARIABLE
        5-DELETE A VARIABLE
        6-PERFORM REGRESSION COMPUTATIONS
        7-STUDY ANOTHER MODEL
        8-STORE DATA ON DISK
        9-WEIGHT DATA
       10-TRANSFORM DATA
       11-QUIT
```

We are now ready to perform the regression computations by specifying Option 6, PERFORM REGRESSION COMPUTATIONS. At this point, the program provides three options. Options 1 and 2 are the regression techniques described earlier in this chapter. Option 3, DATA MANAGEMENT OPTIONS, returns control to the 11 data management options we have been using.

```
OPTION ? 6

      AVAILABLE OPTIONS:

        1-MULTIPLE REGRESSION
        2-STEPWISE MULTIPLE REGRESSION
        3-DATA MANAGEMENT OPTIONS
```

Because we have only one independent variable, stepwise regression is not needed; therefore, we will choose option 1, MULTIPLE REGRESSION. We would also like to see each iteration of the regression computatons displayed at our CRT.

```
OPTION ? 1

PRINT SOLUTION AT EACH ITERATION (Y/N)? Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? C

        MULTIPLE REGRESSION
        ===================

REGRESSION NUMBER 1

   SALES = 8366.66667
COEFFICIENT OF DETERMINATION = 0
STD DEVIATION OF ESTIMATE = 315.6369
```

```
          FINAL SOLUTION

REGRESSION NUMBER 2

    SALES = 4251.94936
  +   1.21378092 ORDERS
COEFFICIENT OF DETERMINATION = .8369922
STD DEVIATION OF ESTIMATE = 142.4777
```

Regression iteration number 1 represents averaging the SALES data. The final solution, regression iteration number 2, is the best fit regression equation using ORDERS to estimate SALES. The coefficient of determination is .837; thus, 83.7% of the variation in SALES can be explained by the ORDERS.

After pressing return to continue, we are asked if we want to see a comparison of the actual and predicted values for SALES. In addition, we must specify whether the program-supplied T-statistics are to be used to calculate the confidence limits on the SALES prediction. After responding to these prompts, the remaining regression results are displayed. Looking at the confidence limits on the predicted values, we can see that the forecast for period seven is 8742.939, and that the lower and upper 95% confidence limits are 8257.523 and 9228.355, respectively. Thus, we are 95% sure that the SALES for period seven will be within this range, and the expected value is 8742.939.

```
PRESS RETURN TO CONTINUE ->

WANT TO SEE LIMITS ON PREDICTIONS (Y/N)?Y
USE PROGRAM SUPPLIED T-STATISTICS
FOR LIMITS (Y/N) ? Y

ACTUAL VERSUS PREDICTED VALUES FOR SALES

ACTUAL          PREDICTED          DIFFERENCE
-------------------------------------------------
 8350           8560.872          -210.87162
 8730           8621.561           108.43934
 8120           8196.737          - 76.73734
 8450           8318.115           131.88457
 7900           7893.293             6.70752
 8650           8609.422            40.57715

PRESS RETURN TO CONTINUE ->

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 4

T-STATISTIC = 2.776

LOWER LIMIT          PREDICTED          UPPER LIMIT
-------------------------------------------------
8117.41014          8560.87162          9004.3331
8166.71522          8821.56067          9076.40611
7757.03125          8196.73734          8636.44343
7889.87305          8318.11543          8746.35782
7376.97455          7893.29211          8409.60967
8157.07563          8609.42285          9061.77008
8257.5229           8742.93875          9228.35461

PRESS RETURN TO CONTINUE ->
```

Next, we might see how these results would look if they had been displayed on the printer. The following information will appear on the CRT:

```
AVAILABLE OPTIONS:

   1-MULTIPLE REGRESSION

   2-STEPWISE MULTIPLE REGRESSION

   3-DATA MANAGEMENT OPTIONS

OPTION ? 1

PRINT SOLUTION AT EACH ITERATION (Y/N)?Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)?P

          MULTIPLE REGRESSION
         ====================

WANT TO SEE LIMITS ON PREDICTION (Y/N)?Y

USE PROGRAM SUPPLIED T-STATISTICS
FOR LIMITS (Y/N) ? Y
```

Then the regression results would be printed at the printer as follows:

```
           MULTIPLE REGRESSION
          ====================

REGRESSION NUMBER 1
     SALES = 8366.66667
COEFFICIENT OF DETERMINATION = 0
STD DEVIATION OF ESTIMATE = 315.636936
     FINAL SOLUTION

REGRESSION NUMBER 2
     SALES = 4251.94936
  + 1.21378092 ORDERS
COEFFICIENT OF DETERMINATION = .8369922
STD DEVIATION OF ESTIMATE = 142.477744

ACTUAL      PREDICTED         DIFFERENCE          %DIFFERENCE
 8350      8560.87162       -210.871616          -2.52540858
 8730      8621.56067        108.439339           1.24214592
 8120      8196.73734        -76.73739           - .945041121
 8450      8318.11543        131.884567           1.56076411
 7900      7893.29211          6.70788956           .0849099944
 8650      8609.42285         40.5771484           .469099982

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 4
```

```
T-STATISTIC = 2.776

LOWER LIMIT              PREDICTED              UPPER LIMIT
-----------------------------------------------------------
8117.41041               8560.87162             9004.3331
8166.71522               8621.56067             9076.40611
7757.03125               8196.73734             8636.44343
7889.87305               8318.11543             8746.35782
7376.97455               7893.29211             8409.60967
8157.07563               8609.42285             9061.77008
8257.5229                8742.93875             9228.35461
```

Since we have a solution to Example 2, we will now use data management option 11, QUIT, to end program execution.

EXAMPLE 3. Assume that another month has passed at the Brady Company and that sales totaled $8810 for the seventh month. Orders totaling $3760 were on hand at the beginning of the eighth month. Develop a sales forecast for the eighth month using the Multiple and Stepwise Regression Program.

SOLUTION. The data used for the seventh month sales forecast were stored on a disk file named BRASALES. Therefore, it will be easy to use these data to develop a new forecast. We need only to designate that the data be input from a disk and then enter the name of the disk file containing the data. After this, we will input the newly obtained data for the seventh month.

```
RUN

**************************************
MULTIPLE AND STEPWISE REGRESSION PROGRAM
**************************************

LIST PROGRAM OPTIONS (Y/N) ? N

HOW ENTER DATA-KEYBOARD OR DISK (K/D)? D
ENTER NAME OF DISK:FILE ? BRASALES

     DATA MANAGEMENT OPTIONS:
     ========================

          1-LIST DATA
          2-CORRECT DATA
          3-ADD TO DATA
          4-ADD AN INDEPENDENT VARIABLE
          5-DELETE A VARIABLE
          6-REGRESSION COMPUTATIONS
          7-STUDY ANOTHER MODEL
          8-STORE DATA ON DISK
          9-WEIGHT DATA
         10-TRANSFORM DATA
         11-QUIT
```

We need to add the newly obtained data (sales for the seventh month and orders for the eighth month) to the data that was input from the disk file. Using option 3, ADD TO DATA, we can do this.

```
OPTION ? 3

FORECAST REQUIRED (Y/N)? Y

NUMBER OF PERIODS TO BE FORECAST? 1

ENTER NUMBER DATA POINTS FOR EACH
INDEPENDENT   VARIABLE INCLUDING POINTS
USED FOR FORECAST ? 8

ENTER ADDITIONAL DATA FOR INDEPENDENT
VARIABLE ORDERS
VALUE FOR POINT 8?3760

ENTER ADDITIONAL DATA FOR DEPENDENT
VARIABLE SALES
VALUE FOR POINT ??8810

DATA MANAGEMENT OPTIONS:
 ======================
          1-LIST DATA
          2-CORRECT DATA
          3-ADD TO DATA
          4-ADD AN INDEPENDENT VARIABLE
          5-DELETE A VARIABLE
          6-REGRESSION COMPUTATIONS
          7-STUDY ANOTHER MODEL
          8-STORE DATA ON DISK
          9-WEIGHT DATA
         10-TRANSFORM DATA
         11-QUIT
```

Having added the additional data, we perform the regression computations resulting in a forecast for the eighth month of $8841.23 total sales.

```
OPTION 6

     AVAILABLE OPTIONS:

          1-MULTIPLE REGRESSION
          2-STEPWISE MULTIPLE REGRESSION
          3-DATA MANAGEMENT OPTIONS

OPTION ? 1
PRINT SOLUTION AT EACH ITERATION (Y/N)? N

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? C

     MULTIPLE REGRESSION
    ====================

        FINAL SOLUTION

REGRESSION NUMBER 2
```

```
     SALES = 4094.03983
 +    1.2625508 ORDERS
COEFFICIENT OF DETERMINATION = .8737094
STD DEVIATION OF ESTIMATE = 129.7577

PRESS RETURN TO CONTINUE->

WANT TO SEE LIMITS ON PREDICTION (Y/N)? Y
USE PROGRAM SUPPLIED T-STATISTICS
FOR LIMITS (Y/N) ? Y

ACTUAL VERSUS PREDICTED VALUES FOR SALES
ACTUAL                PREDICTED                DIFFERENCE
-----------------------------------------------------------
 8350                 8576.096                -226.09517
 8730                 8639.223                  90.7772999
 8120                 8197.33                 - 77.3299299
 8450                 8323.585                 126.415
 7900                 7881.692                  18.30778
 8650                 8626.597                  23.40281
 8810                 8765.478                  44.52222

PRESS RETURN TO CONTINUE ->

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 5

T-STATISTIC = 2.571

LOWER LIMIT              PREDICTED              UPPER LIMIT
-----------------------------------------------------------
8213.78126              8576.09517              8938.40907
8271.04125              8639.2227               9007.40416
7826.46943              8197.32993              8568.19043
7963.92306              8323.5801               8683.24695
7451.99189              7881.69223              8311.39256
8259.74778              8626.5972               8993.44661
8379.86259              8765.47778              9151.09298
8441.84609              8841.231083             9240.61558

PRESS RETURN TO CONTINUE ->

  AVAILABLE OPTIONS:

        1-MULTIPLE REGRESSION
        2-STEPWISE MULTIPLE REGRESSION
        3-DATA MANAGEMENT OPTIONS
```

Looking back at the comparison of actual versus predicted values, we see that our predictions have been better for the more recent months. Consequently, we might try weighting these later periods more heavily. When the data is weighted, we should make sure that the average weight is one. Therefore, the sum of the weights should total seven since we will be weighting seven data elements for each variable (ORDERS and SALES for periods 1 through 7). Using option 9, WEIGHT DATA, we can weight the data however we want. The user should note that these weights are retained until one of the

following actions occur: (1) program execution is terminated by specifying option 11, QUIT; (2) weights are changed using option 9, WEIGHT DATA; or (3) option 7, STUDY ANOTHER MODEL is specified.

```
OPTION? 3

    DATA MANAGEMENT OPTIONS:
    ========================

          1-LIST DATA
          2-CORRECT DATA
          3-ADD TO DATA
          4-ADD AN INDEPENDENT VARIABLE
          5-DELETE A VARIABLE
          6-REGRESSION COMPUTATIONS
          7-STUDY ANOTHER MODEL
          8-STORE DATA ON DISK
          9-WEIGHT DATA
         10-TRANSFORM DATA
         11-QUIT

OPTION ? 9
ENTER WEIGHT FOR DATA POINT 1?.5
ENTER WEIGHT FOR DATA POINT 2?.5
ENTER WEIGHT FOR DATA POINT 3?1
ENTER WEIGHT FOR DATA POINT 4?1
ENTER WEIGHT FOR DATA POINT 5?1
ENTER WEIGHT FOR DATA POINT 6?1.5
ENTER WEIGHT FOR DATA POINT 7?1.5

    DATA MANAGEMENT OPTIONS:
    ========================

          1-LIST DATA
          2-CORRECT DATA
          3-ADD TO DATA
          4-ADD AN INDEPENDENT VARIABLE
          5-DELETE A VARIABLE
          6-REGRESSION COMPUTATIONS
          7-STUDY ANOTHER MODEL
          8-STORE DATA ON DISK
          9-WEIGHT DATA
         10-TRANSFORM DATA
         11-QUIT
```

Having weighted the data, we might list them to see if any changes have occurred. We will see that nothing has been modified, because the weights are applied during the regression computations. Even then, the original data are maintained unaltered while the weighted data are temporarily stored to be used during the regression computations.

```
OPTION? 1

ENTER NAME OF VARIABLE IDENTIFYING DATA
TO BE LISTED? SALES
```

```
LIST OF DATA FOR SALES:

 8350
 8730
 8120
 8450
 7900
 8650
 8810

LIST ADDITIONAL DATA (Y/N)? Y

ENTER NAME OF VARIABLE IDENTIFYING
DATA TO BE LISTED ? ORDERS

LIST OF DATA FOR ORDERS:
========================

 3550
 3600
 3250
 3350
 3000
 3590
 3700
 3760

LIST ADDITIONAL DATA (Y/N) ? N

   DATA MANAGEMENT OPTIONS:
         1-LIST DATA
         2-CORRECT DATA
         3-ADD TO DATA
         4-ADD AN INDEPENDENT VARIABLE
         5-DELETE A VARIABLE
         6-REGRESSION COMPUTATIONS
         7-STUDY ANOTHER MODEL
         8-STORE DATA ON DISK
         9-WEIGHT DATA
        10-TRANSFORM DATA
        11-QUIT
```

We can perform the regression computations at this point to see if the data weights have improved the fit of the regression equation. The results show that the coefficient of determination is better and that the prediction for the last four periods is better. Using the new regression equation,

$$\text{SALES} = 4000.563 + 1.293905(\text{ORDERS}),$$

we obtain a sales forecast of \$8865.65. This figure is slightly higher than the forecast made using the unweighted data.

```
OPTION? 6

  AVAILABLE OPTIONS:

         1-MULTIPLE REGRESSION
         2-STEPWISE MULTIPLE REGRESSION
         3-DATA MANAGEMENT OPTIONS
```

```
OPTION ? 1

PRINT SOLUTION AT EACH ITERATION (Y/N)? N

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? C

      MULTIPLE REGRESSION
      ===================

          FINAL SOLUTION

REGRESSION NUMBER 2

      SALES = 4000.56358
  +    1.29390522 ORDERS
COEFFICIENT OF DETERMINATION = .9240319
STD DEVIATION OF ESTIMATE = 103.8366

PRESS RETURN TO CONTINUE ->

WANT TO SEE LIMITS ON PREDICTIONS (Y/N)?Y
USE PROGRAM SUPPLIED T-STATISTICS
FOR LIMITS (Y/N)? Y

ACTUAL VERSUS PREDICTED VALUES FOR SALES
ACTUAL                PREDICTED                DIFFERENCE
 8350                  8593.927                -243.92712
 8730                  8658.622                  71.3776199
 8120                  8205.756                - 85.7555499
 8450                  8335.146                 114.85392
 7900                  7882.279                  17.72075
 8650                  8645.683                   4.31667
 8810                  8788.013                  21.98709

PRESS RETURN TO CONTINUE ->

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 5

T-STATISTIC = 2.571

LOWER LIMIT           PREDICTED                UPPER LIMIT
-----------------------------------------------------------
8304.99007            8593.92713               8882.86418
8365.59825            8658.62239               8951.64652
7908.56935            8205.75556               8502.94176
8046.92965            8335.14600               8623.36251
7539.68804            7882.27925               8224.87046
8353.59661            8645.68333               8937.77006
8482.4812             8788.01291               9093.54461
8550.071              8865.64722               9181.22345

PRESS RETURN TO CONTINUE ->

   AVAILABLE OPTIONS:

        1-MULTIPLE REGRESSION
        2-STEPWISE MULTIPLE REGRESSION
        3-DATA MANAGEMENT OPTIONS
```

At this point, it would be interesting to see if the same results would be obtained using the stepwise multiple regression option. After doing this, you can see that the results are the same. Remember that the data weights are still being applied.

```
OPTION? 2

PRINT SOLUTION AT EACH ITERATION (Y/N)? N

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? C

   STEPWISE MULTIPLE REGRESSION

         FINAL SOLUTION

REGRESSION NUMBER 2

      SALES = 4000.56358
  +   1.29390522 ORDERS
COEFFICIENT OF DETERMINATION = .924019
STD DEVIATION OF ESTIMATE = 103.8366

PRESS RETURN TO CONTINUE ->

WANT TO SEE LIMITS ON PREDICTION (Y/N)?N

ACTUAL VERSUS PREDICTED VALUES FOR SALES
ACTUAL              PREDICTED              DIFFERENCE
-------------------------------------------------------
 8350                8593.927              -243.92712
 8730                8658.622               71.3776199
 8120                8205.756              - 85.7555499
 8450                8335.146               144.85392
 7900                7882.279               17.72075
 8650                8645.683                4.31667
 8810                8788.013               21.98709

   AVAILABLE OPTIONS:

         1-MULTIPLE REGRESSION
         2-STEPWISE MULTIPLE REGRESSION
         3-DATA MANAGEMENT OPTIONS

OPTION? 3

   DATA MANAGEMENT OPTIONS:
   ========================

         1-LIST DATA
         2-CORRECT DATA
         3-ADD TO DATA
         4-ADD AN INDEPENDENT VARIABLE
         5-DELETE A VARIABLE
         6-REGRESSION COMPUTATIONS
         7-STUDY ANOTHER MODEL
         8-STORE DATA ON DISK
         9-WEIGHT DATA
```

```
          10-TRANSFORM DATA
          11-QUIT

OPTION ? 11

          ***END OF PROGRAM***
```

EXAMPLE 4. Data have been collected from a laboratory experiment. The rate of growth of a bacteria culture is being studied. The population density in parts per million (D,P/M), as a function of time, is tabulated below:

Days	.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
D,P/M	.3	.7	2.0	5.4	14.8	40.3	109.7	298.1

Previous work has established that the rate of bacteria growth should conform to an equation such as:

$$\text{density} = ae^{bt}.$$

Use the multiple and stepwise regression program to develop a regression equation that relates the bacteria density to time.

SOLUTION. We can use linear regression to fit a line to this data if the data is first transformed, giving

$$\ln(\text{density}) = \ln(a) + bt$$

Because our regression program can perform a natural log transformation, we will let the computer perform the required transformations.

```
RUN

************************************
MULTIPLE AND STEPWISE REGRESSION PROGRAM
************************************

LIST PROGRAM OPTIONS (Y/N) ? N

HOW ENTER DATA-KEYBOARD OR DISK (K/D)? K

ENTER NAME OF DEPENDENT VARIABLE? BACTERIA

ENTER NUMBER OF INDEPENDENT VARIABLES ?1
ENTER NAME OF INDEPENDENT VARIABLE: 1
?TIME

FORECAST REQUIRED (Y/N) ? N
ENTER NUMBER OF DATA POINTS FOR EACH
INDEPENDENT VARIABLE ? 8
```

The raw data (untransformed) will be entered. Then using option 10, TRANSFORM DATA, we will transform the set of data named BACTERIA (the dependent variable).

```
ENTER DATA


8 VALUES REQUIRED FOR INDEPENDENT
```

```
VARIABLE TIME

VALUE FOR POINT 1?.5
VALUE FOR POINT 2?1.
VALUE FOR POINT 3?1.5
VALUE FOR POINT 4?2.
VALUE FOR POINT 5?2.5
VALUE FOR POINT 6?3.
VALUE FOR POINT 7?3.5
VALUE FOR POINT 8?4.

8 VALUES REQUIRED FOR DEPENDENT
VARIABLE BACTERIA
VALUE FOR POINT 1?.3
VALUE FOR POINT 2?.7
VALUE FOR POINT 3?2.
VALUE FOR POINT 4?5.4
VALUE FOR POINT 5?14.8
VALUE FOR POINT 6?40.3
VALUE FOR POINT 7?109.7
VALUE FOR POINT 8?298.1

   DATA MANAGEMENT OPTIONS:
   ========================

        1-LIST DATA
        2-CORRECT DATA
        3-ADD TO DATA
        4-ADD AN INDEPENDENT VARIABLE
        5-DELETE A VARIABLE
        6-REGRESSION COMPUTATIONS
        7-STUDY ANOTHER MODEL
        8-STORE DATA ON DISK
        9-WEIGHT DATA
       10-TRANSFORM DATA
       11-QUIT

OPTION ? 10

   TRANSFORMS AVAILABLE:

      N=NO TRANSFORMS
      E=LOG(X) TO BASE E
      P=POLYNOMIAL X^N

ENTER TRANSFORM CODE FOR VARIABLE  TIME?N
ENTER TRANSFORM CODE FOR VARIABLE  BACTERIA?E
```

Having performed the transformation, we will list the data (using option 1) to see the results. You should note that the program has replaced the original data with the transformed data. Therefore, if we were to store this data in a disk file for later use, we should not transform the data again.

```
    DATA MANAGEMENT OPTIONS:
    ========================

          1-LIST DATA
          2-CORRECT DATA
          3-ADD TO DATA
          4-ADD AN INDEPENDENT VARIABLE
          5-DELETE A VARIABLE
          6-REGRESSION COMPUTATIONS
          7-STUDY ANOTHER MODEL
          8-STORE DATA ON DISK
          9-WEIGHT DATA
         10-TRANSFORM DATA
         11-QUIT

OPTION ? 1

ENTER NAME OF VARIABLE IDENTIFYING DATA
TO BE LISTED ? BACTERIA

LIST OF DATA FOR BACTERIA:

     -1.2039728
     -  .356674943
        .693147181
       1.68639895
       2.69462718
       3.69635147
       4.69774937
       5.697429

LIST ADDITIONAL DATA (Y/N) ? N

    DATA MANAGEMENT OPTIONS:
    ========================

          1-LIST DATA
          2-CORRECT DATA
          3-ADD TO DATA
          4-ADD AN INDEPENDENT VARIABLE
          5-DELETE A VARIABLE
          6-REGRESSION COMPUTATIONS
          7-STUDY ANOTHER MODEL
          8-STORE DATA ON DISK
          9-WEIGHT DATA
         10-TRANSFORM DATA
         11-QUIT
```

We are ready now to perform the regression computations. Remember that the regression equation obtained by the program will be in terms of the transformed data:

$\ln(\text{density}) = \ln a + bt.$

```
OPTION ? 6
```

```
      AVAILABLE OPTIONS:

            1-MULTIPLE REGRESSION
            2-STEPWISE MULTIPLE REGRESSION
            3-DATA MANAGEMENT OPTIONS

OPTION ? 1

PRINT SOLUTION AT EACH ITERATION (Y/N)? Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? C

          MULTIPLE REGRESSION
         ====================

REGRESSION NUMBER 1
      BACTERIA = 2.20063193
COEFFICIENT OF DETERMINATION = 0
STD DEVIATION OF ESTIMATE = 2.4382

     FINAL SOLUTION

REGRESSION NUMBER 2
      BACTERIA = -2.27792746
   +   1.99047084 TIME
COEFFICIENT OF DETERMINATION = .9997173
STD DEVIATION OF ESTIMATE = .0443

PRESS RETURN TO CONTINUE ->

WANT TO SEE LIMITS ON PREDICTIONS (Y/N)?Y
USE PROGRAM SUPPLIED T-STATISTICS
FOR LIMITS (Y/N)? Y

ACTUAL VERSUS PREDICTED VALUES FOR
BACTERIA
    ACTUAL              PREDICTED              DIFFERENCE
-----------------------------------------------------------
-1.2039728              -1.283                  .0787199999
-  .356674943           -  .287              -  .0692199999
   .693147181             .708              -  .01463
  1.68639895             1.703              -  .01662
  2.69462718             2.698              -3.62E-03
  3.69635147             3.693               2.87E-03
  4.69774937             4.689               9.029999999E-03
  5.697429               5.684                .01347


PRESS RETURN TO CONTINUE ->

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 6
T-STATISTIC = 2.447
```

```
 LOWER LIMIT          PREDICTED          UPPER LIMIT
-------------------------------------------------------
-1.4116479          -1.28269204         -1.5373619
- .40973776         - .287456624        - .165175489
  .590157713          .707778796          .825399878
 1.58779382          1.70301422          1.81823462
 2.58302924          2.69824964          2.81347004
 3.57586398          3.69348506          3.81110614
 4.56643934          4.68872048          4.81100161
 5.55500005          5.6839559           5.81291175

PRESS RETURN TO CONTINUE ->
    AVAILABLE OPTIONS:

       1-MULTIPLE REGRESSION
       2-STEPWISE MULTIPLE REGRESSION
       3-DATA MANAGEMENT OPTIONS

OPTION ? 3

     DATA MANAGEMENT OPTIONS:
     ========================

         1-LIST DATA
         2-CORRECT DATA
         3-ADD TO DATA
         4-ADD AN INDEPENDENT VARIABLE
         5-DELETE A VARIABLE
         6-REGRESSION COMPUTATIONS
         7-STUDY ANOTHER MODEL
         8-STORE DATA ON DISK
         9-WEIGHT DATA
        10-TRANSFORM DATA
        11-QUIT

OPTION ? 11

     ***END OF PROGRAM***
```

Looking at the coefficient of determination (.9997) for the final solution, we can see that a good fit was obtained. The regression equation is

$$\text{BACTERIA} = -2.277927 + 1.990471(\text{TIME})$$

Assume that TIME = 3 days. The predicted density of the bacteria culture expressed as a natural log would be:

$$\begin{aligned}\ln(\text{BACTERIA}) &= -2.277927 + (1.990471)\,(3)\\ &= -2.277927 + 5.971413\\ &= 3.693486\end{aligned}$$

Removing the log from the above expression, the predicted bacteria density is:

$$\begin{aligned}\text{D,P/M} &= \exp\,(3.693486)\\ &= 40.18487\end{aligned}$$

This figure is very close to the observed value of 40.3. We can write the above regression equation without the natural log. Since

$$e^{-2.77927} = .1024964,$$

then

$$D,P/M = .1024964\ e\ (1.990471)\ (TIME)$$

Consequently, we have developed an equation to predict the density of the bacteria culture.

EXAMPLE 5. Some experimental data have been collected and presented below in tabular form. The dependent variable is Y and X is the independent variable.

X	0	1	2	3	4	5	6	7	8	9
Y	50	65	80	100	130	160	200	245	290	345

Determine if an equation of the form:

$$Y = A + B_1X + B_2X^2 + B_3X^3$$

will provide a good fit.

SOLUTION. The multiple and stepwise regression program will be utilized. Three independent variables will be used:

X1 will represent X,
X2 will represent X^2, and
X3 will represent X^3.

```
RUN

**************************************
MULTIPLE AND STEPWISE REGRESSION PROGRAM
**************************************

LIST PROGRAM OPTIONS (Y/N) ? N

HOW ENTER DATA-KEYBOARD OR DISK (K/D)? K

ENTER NAME OF DEPENDENT VARIABLE ? Y

ENTER NUMBER OF INDEPENDENT VARIABLES ?3

ENTER NAME OF INDEPENDENT VARIABLE 1 :
?X1

ENTER NAME OF INDEPENDENT VARIABLE 2 :
?X2

ENTER NAME OF INDEPENDENT VARIABLE 3 :
?X3

FORECAST REQUIRED (Y/N) ? N

ENTER NUMBER OF DATA POINTS FOR EACH
INDEPENDENT VARIABLE ? 10
```

```
ENTER DATA
```

Since our program can perform a transformation of X, we will let the computer perform the required calculations. Therefore, the values entered for X1, X2, and X3 will be the same.

```
 10 VALUES REQUIRED FOR INDEPENDENT VARIABLE X1

VALUE FOR POINT  1?0
VALUE FOR POINT  2?1
VALUE FOR POINT  3?2
VALUE FOR POINT  4?3
VALUE FOR POINT  5?4
VALUE FOR POINT  6?5
VALUE FOR POINT  7?6
VALUE FOR POINT  8?7
VALUE FOR POINT  9?8
VALUE FOR POINT 10?9

 10 VALUES REQUIRED FOR INDEPENDENT VARIABLE X2

VALUE FOR POINT  1?0
VALUE FOR POINT  2?1
VALUE FOR POINT  3?2
VALUE FOR POINT  4?3
VALUE FOR POINT  5?4
VALUE FOR POINT  6?5
VALUE FOR POINT  7?6
VALUE FOR POINT  8?7
VALUE FOR POINT  9?8
VALUE FOR POINT 10?9

 10 VALUES REQUIRED FOR INDEPENDENT VARIABLE X3

VALUE FOR POINT  1?0
VALUE FOR POINT  2?1
VALUE FOR POINT  3?2
VALUE FOR POINT  4?3
VALUE FOR POINT  5?4
VALUE FOR POINT  6?5
VALUE FOR POINT  7?6
VALUE FOR POINT  8?7
VALUE FOR POINT  9?8
VALUE FOR POINT 10?9

 10 VALUES REQUIRED FOR DEPENDENT VARIABLE Y

VALUE FOR POINT  1?50
VALUE FOR POINT  2?65
VALUE FOR POINT  3?80
VALUE FOR POINT  4?100
VALUE FOR POINT  5?130
VALUE FOR POINT  6?160
VALUE FOR POINT  7?200
VALUE FOR POINT  8?245
```

```
VALUE FOR POINT  9?290
VALUE FOR POINT 10?345

  DATA MANAGEMENT OPTIONS:
  ========================

        1-LIST DATA
        2-CORRECT DATA
        3-ADD TO DATA
        4-ADD AN INDEPENDENT VARIABLE
        5-DELETE A VARIABLE
        6-PERFORM REGRESSION COMPUTATIONS
        7-STUDY ANOTHER MODEL
        8-STORE DATA ON DISK
        9-WEIGHT DATA
       10-TRANSFORM DATA
       11-QUIT
```

Data for variables X2 and X3 will be transformed now.

```
OPTION? 10

     TRANSFORMS AVAILABLE:
           N=NO TRANSFORMS
           E=LOG(X) TO BASE E
           P=POLYNOMIAL X^N

ENTER TRANSFORM CODE FOR VARIABLE X1?N

ENTER TRANSFORM CODE FOR VARIABLE X2?P

ENTER VALUE FOR EXPONENT (X^N) ? 2

ENTER TRANSFORM CODE FOR VARIABLE X3?P

ENTER VALUE FOR EXPONENT (X^N) ? 3

ENTER TRANSFORM CODE FOR VARIABLE Y?N

  DATA MANAGEMENT OPTIONS:
  ========================

      1-LIST DATA
      2-CORRECT DATA
      3-ADD TO DATA
      4-ADD AN INDEPENDENT VARIABLE
      5-DELETE A VARIABLE
      6-REGRESSION COMPUTATIONS
      7-STUDY ANOTHER MODEL
      8-STORE DATA ON DISK
      9-WEIGHT DATA
     10-TRANSFORM DATA
     11-QUIT

OPTION ? 6
```

```
   AVAILABLE OPTIONS:

      1-MULTIPLE REGRESSION
      2-STEPWISE MULTIPLE REGRESSION
      3-DATA MANAGEMENT OPTIONS
```

We will first try multiple regression. This time the output should be displayed at the printer.

```
OPTION?1

PRINT SOLUTION AT EACH ITERATION (Y/N)? Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? P

         MULTIPLE REGRESSION
        ====================

REGRESSION NUMBER  1
     Y = 166.5
COEFFICIENT OF DETERMINATION = 0
STD DEVIATION OF ESTIMATE = 100.858812

REGRESSION NUMBER  2
     Y = 19.6363636
  +   32.6363636 X1
COEFFICIENT OF DETERMINATION = .9598144
STD DEVIATION OF ESTIMATE = 21.4449613

REGRESSION NUMBER 3
     Y = 51.2272728
  +   8.94318181 X1
  +   2.63257776 X2
COEFFICIENT OF DETERMINATION = .9997836
STD DEVIATION OF ESTIMATE = 1.68228683

    FINAL SOLUTION

REGRESSION NUMBER 4
     Y = 51.2027966
  +   8.98795652 X1
  +   2.61946357 X2
  +   9.71273435E-04 X3
COEFFICIENT OF DETERMINATION = .9997836
STD DEVIATION OF ESTIMATE =   1.81694381

WANT TO SEE LIMITS ON PREDICTION (Y/N)?Y
USE PROGRAM SUPPLIED T-STATISTICS
FOR LIMITS (Y/N) ? Y
ACTUAL      PREDICTED       DIFFERENCE        %DIFFERENCE
   50       51.2027966     -1.20279665       -2.40559331
   65       62.811189       2.18881099        3.36740152
```

```
  80      79.6643361      .335663885      .419579856
 100     101.768066     -1.76806569     -1.76806569
 130     129.128205       .871794701      .670611308
 160     161.750583     -1.75058258     -1.09411411
 200     199.641025       .358974755      .179487377
 245     242.8054361     2.19463915       .89577108
 290     291.249417     -1.24941707     - .430833471
 345     344.979022       .0209784508    6.08071037E-03

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 6

T-STATISTIC = 2.447

LOWER LIMIT              PREDICTED              UPPER LIMIT
-----------------------------------------------------------
 45.1985123              51.2027966              57.207081
 57.7387185              62.81189                67.8836595
 74.5443971              79.6643361              84.7842752
 96.6842529             101.768066              106.851878
124.175234              129.128205              134.081177
156.797611              161.750583              166.703554
194.557213              199.641025              204.724838
237.685422              242.805361              247.9253
286.176947              291.249417              296.321888
338.974737              344.979022              350.983306

PRESS RETURN TO CONTINUE ->
```

Look at the final solution and regression number 3. There is no difference in the coefficient of determination. The last variable added to the equation, X3, may not be needed. We will try the stepwise multiple regression option to see if this reasoning is correct.

```
     AVAILABLE OPTIONS:

        1-MULTIPLE REGRESSION
        2-STEPWISE MULTIPLE REGRESSION
        3-DATA MANAGEMENT

OPTION? 2

PRINT SOLUTION AT EACH ITERATION (Y/N)? Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? P

    STEPWISE MULTIPLE REGRESSION
    ============================

REGRESSION NUMBER   1
     Y = 166.5
COEFFICIENT OF DETERMINATION = 0
STD DEVIATION OF ESTIMATE = 100.858812
```

```
REGRESSION NUMBER  2
      Y = 65.2252965
  +    3.553498 X2
COEFFICIENT OF DETERMINATION = .994506
STD DEVIATION OF ESTIMATE = 7.92928127

      FINAL SOLUTION

REGRESSION NUMBER 3
      Y = 51.2272727
  +    8.94318183 X1
  +    2.63257576 X2
COEFFICIENT OF DETERMINATION = .9997836
STD DEVIATION OF ESTIMATE = 1.68228648

WANT TO SEE LIMITS ON PREDICTION (Y/N)?Y
USE PROGRAM SUPPLIED T-STATISTICS
FOR LIMITS (Y/N) ? Y

ACTUAL      PREDICTED       DIFFERENCE       %DIFFERENCE
  50        51.2272727     -1.22727269      -2.45454538
  65        62.8030303      2.19696973       3.37995344
  80        79.6439394       .356060624       .44507578
 100       101.75          -1.75            -1.75
 130       129.121212       .878787875       .675990673
 160       161.757576      -1.75757575      -1.09848484
 200       199.659091       .340909064       .170454532
 245       242.825758      2.17424244        .887445893
 290       291.257576      -1.25757575      - .43364681
 345       344.954545       .0454545021      .013175218

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 7

T-STATISTIC = 2.365

LOWER LIMIT          PREDICTED          UPPER LIMIT
 46.1661745           51.2272727          56.2883708
 58.3038815           62.8030303          67.3021791
 75.3159653           79.6439394          83.9719134
 97.3999161          101.75              106.100084
124.719062           129.121212          133.523362
157.355426           161.757576          166.159726
195.309007           199.659091          204.009175
238.497784           242.825758          247.153732
286.758427           291.257576          295.756725
339.893447           344.954545          350.015644
```

We were right. The final results of the stepwise regression computations left X3 out of the regression equation. Remember that the stepwise procedure will not include an independent variable in the regression unless it significantly reduces the variance of the

results. In most cases, it is not so obvious when a variable should be included in the final equation. Consequently, a stepwise regression program can be very useful.

EXAMPLE 6. Since the stepwise regression procedure will not include variables in the final solution that are not significant, the temptation is to use only this procedure. However, there are times when the multiple regression procedure can produce better results. Consider the following data, where Y is the dependent variable.

X	1	2	3	4	5	6	7	8
Y	2	9	14	16	19	18	13	6

Determine if an equation of the form

$$Y = A + B_1X + B_2X^2$$

will provide a good fit to this data.

SOLUTION. We will show only the results after having entered the data. The data associated with X2 was transformed, and then the stepwise regression option was specified.

```
OPTION? 10

   TRANSFORMS AVAILABLE:

        N = NO TRANSFORMS
        E = LOG(X) TO BASE E
        P = POLYNOMIAL X^N

ENTER TRANSFORM CODE FOR VARIABLE   X1?N

ENTER TRANSFORM CODE FOR VARIABLE   X2?P

ENTER VALUE FOR EXPONENT (X^N) ? 2

ENTER TRANSFORM CODE FOR VARIABLE Y?N

   DATA MANAGEMENT OPTIONS:
   ------------------------
   ------------------------

         1-LIST DATA
         2-CORRECT DATA
         3-ADD TO DATA
         4-ADD AN INDEPENDENT VARIABLE
         5-DELETE A VARIABLE
         6-REGRESSION COMPUTATIONS
         7-STUDY ANOTHER MODEL
         8-STORE DATA ON DISK
         9-WEIGHT DATA
        10-TRANSFORM DATA
        11-QUIT

OPTION ? 6

     AVAILABLE OPTIONS:
```

```
        1-MULTIPLE REGRESSION
        2-STEPWISE MULTIPLE REGRESSION
        3-DATA MANAGEMENT OPTIONS

OPTION? 2

PRINT SOLUTION AT EACH ITERATION (Y/N)?Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)?C

          STEPWISE MULTIPLE REGRESSION
          ============================

                    FINAL SOLUTION

REGRESSION NUMBER 1
      Y = 12.125
COEFFICIENT OF DETERMINATION = 0
STD DEVIATION OF ESTIMAE = 5.9866

PRESS RETURN TO CONTINUE ->

WANT TO SEE LIMITS ON PREDICTIONS (Y/N)? Y
USE PROGRAM SUPPLIED T-STATISTICS
LIMITS (Y/N) ? Y

ACTUAL VERSUS PREDICTED VALUES FOR Y
ACTUAL          PREDICTED          DIFFERENCE
   2              12.125           -10.125
   9              12.125           - 3.125
  14              12.125             1.875
  16              12.125             3.875
  19              12.125             6.875
  18              12.125             5.875
  13              12.125              .874999999
   6              12.125           - 6.125

PRESS RETURN TO CONTINUE ->

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 7

T-STATISTIC = 2.365

 LOWER LIMIT          PREDICTED          UPPER LIMIT
-2.89213481           12.125             27.1421348
-2.89213481           12.125             27.1421348
-2.89213481           12.125             27.1421348
-2.89213481           12.125             27.1421348
-2.89213481           12.125             27.1421348
-2.89213481           12.125             27.1421348
-2.89213481           12.125             27.1421348
-2.89213481           12.125             27.1421348
```

The results show that there is no correlation. Consequently, the average value for the dependent variable is used to predict any other value for Y. Now, we will try the multiple regression option.

```
      AVAILABLE OPTIONS:

         1-MULTIPLE REGRESSION
         2-STEPWISE MULTIPLE REGRESSION
         3-DATA MANAGEMENT OPTIONS

OPTION ? 1

PRINT SOLUTION AT EACH ITERATION (Y/N)? Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? C

          MULTIPLE REGRESSION
         ====================

REGRESSION NUMBER  1
      Y = 12.125
COEFFICIENT OF DETERMINATION = 0
STD DEVIATION OF ESTIMATE = 5.9866

REGRESSION NUMBER 2
      Y = 8.75
  +    .75 X1
COEFFICIENT OF DETERMINATION = .0941704
STD DEVIATION OF ESTIMATE = 6.1543

           FINAL SOLUTION

REGRESSION NUMBER  3
      Y = -8.48214216
  +    11.0892853 X1
  +    -1.14880948
COEFFICIENT OF DETERMINATION = .9779581
STD DEVIATION OF ESTIMATE = 1.0516

PRESS RETURN TO CONTINUE ->

WANT TO SEE LIMITS ON PREDICTIONS (Y/N)?Y
USE PROGRAM SUPPLIED T-STATISTICS
FOR LIMITS (Y/N)? Y

ACTUAL VERSUS PREDICTED VALUES FOR Y
ACTUAL              PREDICTED              DIFFERENCE
------------------------------------------------------
   2                  1.458                  .541669999
   9                  9.101                - .10119
  14                 14.446                - .44643
  16                 17.494                -1.49405
  19                 18.244                  .755949999
  18                 16.696                 1.30357
  13                 12.851                  .14881
   6                  6.708                - .708329999

PRESS RETURN TO CONTINUE ->
```

```
CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 5
T-STATISTIC = 2.571

 LOWER LIMIT              PREDICTED              UPPER LIMIT
-------------------------------------------------------------
- 2.07558826              1.45833367              4.99225559
  6.04250215              9.10119053             12.1598789
 11.4451853              14.4464284              17.4476716
 14.435359               17.4940474              20.5527358
 15.185359               18.2440474              21.3027358
 13.6951853              16.6964384              19.6976715
  9.79250211             12.8511905              15.9098789
  3.17441172              6.70833364             10.2422556
```

Notice what happened. Using X1 and X2 together results in a regression equation having a very good fit. The coefficient of determination has a value of .9779. However, using X1 and X2 alone would result in equations having coefficients of determination less than .1.

EXAMPLE 7. One assumption made in linear regression is that the independent variables are independent of each other. If one of the independent variables can be expressed as a linear combination of the others, the multiple regression computations cannot be completed. However, the stepwise regression computations can be performed because this procedure will not consider dependent variables. To illustrate this, consider the following data where Y is the dependent variable and X2 is the same as X1 (X1 and X2 are not independent):

X1	1	2	3	4	5	6	7	8
X2	1	2	3	4	5	6	7	8
Y	10	20	30	40	50	60	70	80

Use the multiple and stepwise regression program to develop a regression equation using this data.

SOLUTION. Only the regression computation results will be shown since we understand how to input the data.

```
OPTION? 6

      AVAILABLE OPTIONS:
            1-MULTIPLE REGRESSION
            2-STEPWISE MULTIPLE REGRESSION
            3-DATA MANAGEMENT OPTIONS

OPTION? 1
PRINT SOLUTION AT EACH ITERATION (Y/N)? Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? C

SOLUTION CANNOT BE FOUND USING MULTIPLE
REGRESSION.   VARIABLES ARE NOT INDEPENDENT.
TRY THE STEPWISE REGRESSION OPTION.
```

```
      AVAILABLE OPTIONS:
            1-MULTIPLE REGRESSION
            2-STEPWISE MULTIPLE REGRESSION
            3-DATA MANAGEMENT OPTIONS
```

Note the message which appeared. The multiple regression procedure could not find a final solution because of the dependence between X1 and X2. We will now try the stepwise regression option.

```
OPTION? 2
PRINT SOLUTION AT EACH ITERATION (Y/N)? Y

OUTPUT TO APPEAR AT CRT OR PRINTER (C/P)? C


            STEPWISE MULTIPLE REGRESSION


REGRESSION NUMBER  1
      Y = 45
COEFFICIENT OF DETERMINATION = 0
STD DEVIATION OF ESTIMATE = 24.4949

FINAL SOLUTION

REGRESSION NUMBER 2
      Y = 0
  +    10 X1
COEFFICIENT OF DETERMINATION = 1
STD DEVIATION OF ESTIMATE = 0

PRESS RETURN TO CONTINUE?

WANT TO SEE LIMITS ON PREDICTIONS (Y/N)>? Y
USE PROGRAM SUPPLIED T-STATISTICS FOR LIMITS (Y/N)? Y
ACTUAL VERSUS PREDICTED VALUES FOR Y

ACTUAL                PREDICTED              DIFFERENCE
  10                    10                     0
  20                    20                     0
  30                    30                     0
  40                    40                     0
  50                    50                     0
  60                    60                     0
  70                    70                     0
  80                    80                     0

PRESS RETURN TO CONTINUE

CONFIDENCE LIMITS ON PREDICTED VALUES

DEGREES OF FREEDOM = 6   T-STATISTIC = 2.447
```

LOWER LIMIT	PREDICTED	UPPER LIMIT
10	10	10
20	20	20
30	30	30
40	40	40
50	50	50
60	60	60
70	70	70
80	80	80

The results confirm the above remarks. A solution is found by eliminating X2 from the final results.

The above examples illustrate how to use the Multiple and Stepwise Regression Program. These examples also illustrate why it is good to have both procedures (multiple linear regression and a stepwise linear regression) available.

4.4 Summary

Linear Regression, a widely used curve fitting technique, was discussed in this chapter. Simple linear regression was first presented and a program that uses this technique was provided. Next, stepwise regression was discussed. Several examples were used to explain a program that incorporates two techniques: multiple linear regression and stepwise linear regression. Curve fitting is a common task; as a result, we made these programs easy to use by including data management capabilities within the programs. Space did not permit an in-depth discussion of applied regression analysis; however, Ralston (1964) provides an excellent discussion of this topic.

References

Draper, N. R., and H. Smith, *Applied Regression Analysis*, John Wiley, New York, 1966.

Ralston, Anthony and Herbert S. Wilf, *Mathematical Methods for Digital Computers Volume 1*, John Wiley, New York, 1964.

Exercises

1. A series of experiments has been performed applying various amounts of a reagent to a chemical solution to obtain a precipitate of a particular substance. The following data have been collected:

Precipitate	Reagent
20.8	6.7
25.2	9.2
16.4	4.3
22.5	8.1
18.9	7.9
21.6	7.1

   Fit a straight line to this data using the reagent as the independent variable.

2. A tire manufacturer has been testing a new tire design. One test involves the stopping distance (reaction plus braking distance) of a 2400 pound automobile at various speeds. From this test, some data have been collected:

Stopping Distance (Ft)	Miles Per Hour
41	15
53	25
88	35
130	45
192	55
268	65

Fit a curve to this data, then estimate the stopping distance at 60 miles per hour.

3. It is thought that the salary of an engineer or scientist is related to the number of years of experience. To test this hypothesis, the following data was collected from a random sample of employees at one company.

Annual Salary	Number of Years of Experience
$25,000	6
17,000	1
35,000	10
21,000	4
20,000	2
30,000	8

Estimate the relationship of annual salary to years of experience.

4. In the production of semiconductors, the average yield (percent good devices) per lot increases as more lots are produced. During the past year a firm started producing a new memory device. Data describing the average yield per lot has been collected:

Average Yield Per Lot (%)	Cumulative Number of Lots Produced
20	10
36	20
49	40
60	80
68	160
74	320

Using linear regression, predict what the average yield will be for lot number 400.

5. The viscosity of a liquid varies with temperature. Fit a curve to the following data:

Viscosity	Temperature
252	5
228	15
204	25
188	35
176	45
160	55

6. A series of tests were made and the following data was collected:

Y	X1	X2
34	32	34.3
78	36	53.4
13	30	16.1
55	33	23.9
48	31	42.6
24	27	27.1
24	29	35.3

Develop a regression equation using this data.

7. Using the following data, develop a least squares line:

Y	X1	X2	X3
71	53	25	15
57	44	30	11
43	36	40	7
89	65	15	19
32	32	50	5
66	49	27	13
54	41	32	10
72	55	20	16
38	34	45	6

Are all three independent variables needed in the regression equation?

8. Speciality Engineering is a very successful company. Historical annual earnings per share and corresponding end-of-year stock prices are given below. Management estimates that this year's earnings will be $.83 per share. Using this data, you are to predict the end-of-year stock price.

Year	Stock Price	Earnings per Share
1	$1.52	$.02
2	2.07	.04
3	1.24	.02
4	3.25	.06
5	7.72	.15
6	12.17	.37
7	20.53	.54
8	31.41	.65
9	52.98	.96

5

Solution of Simultaneous Linear Equations

A set of simultaneous linear equations represents a set of conditions that must be satisfied in a particular situation. The solution is a set of variables, $x_1, x_2, \ldots, x_n$, whose values satisfy all equations simultaneously. The equations may be linear or nonlinear, although we will only examine the linear case.

We present both analytic and numerical solution procedures in this chapter. The analytic procedures, matrix inverse method and Gauss-Jordan method, are based on matrix operations which were developed in Chapter 3. A numerical iterative procedure, the Gauss-Seidel Method, also will be presented.

5.1 Simultaneous Linear Equations—General Form

$$
\begin{aligned}
a_{11}\,x_1 + a_{12}\,x_2 + \cdots + a_{1n}\,x_n &= b_1 \\
a_{21}\,x_1 + a_{22}\,x_2 + \cdots + a_{2n}\,x_n &= b_2 \\
&\vdots \\
a_{m1}\,x_1 + a_{m2}\,x_2 + \cdots + a_{mn}\,x_n &= b_m
\end{aligned}
$$

This can be represented in matrix form as:

$$A\vec{x} = \vec{b}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

If $m \neq n$, that is, if the number of equations is not equal to the number of unknowns, then several situations could exist. There could be no solution to the set of equations, a unique solution, or an infinite number of solutions. One or more equations could also be "redundant," or implied by other equations. The determination of these conditions is based upon linear algebra concepts which we will not discuss here. Our discussion will be limited to the case $m = n$. If you want information about the $m \neq n$ case, see Schmidt and Davis (1981) or Hadley (1961).

Another interesting situation occurs when all the b_i, $i = 1, \ldots, n$, are zero. In this case, the set of equations is said to be homogeneous. An obvious solution in this instance, called the trivial solution, is $\vec{x} = 0$. This is generally not the solution of interest. We will not examine the special case of homogeneous equations here, but you can study it further by referring to Hadley (1961).

The analytic procedures presented in this chapter use matrix concepts that were introduced in Chapter 3. You may wish to review that material before continuing in this chapter. A numerical procedure, the Gauss-Seidel method, also will be presented. Although not appropriate in all circumstances, this is a useful alternative to analytic techniques.

5.2 Matrix Inverse Method

By examining the matrix representation of the set of equations, we see that the solution for $\vec{x}$ can be found by premultiplying the right hand-side vector $\vec{b}$ by the inverse of the matrix A.

$$\underline{A}\,\vec{x} = \vec{b}$$

$$\underline{A}^{-1}\underline{A}\,\vec{x} = \underline{A}^{-1}\vec{b}$$

$$\underline{I}\,\vec{x} = \underline{A}^{-1}\vec{b}$$

$$\vec{x} = \underline{A}^{-1}\vec{b}$$

The following example illustrates the solution procedure.

EXAMPLE 1. Solve the following set of simultaneous linear equations for $\vec{x}$.

$$\begin{aligned} 3x_1 + 2x_2 + 2x_3 &= 8 \\ x_2 + x_3 &= 4 \\ x_1 + 3x_3 &= 5 \end{aligned}$$

SOLUTION. First, let's specify the matrix $\underline{A}$ and vector $\vec{b}$.

$$\underline{A} = \begin{bmatrix} 3 & 2 & 2 \\ 0 & 1 & 1 \\ 1 & 0 & 3 \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} 8 \\ 4 \\ 5 \end{bmatrix}$$

We find $\underline{A}^{-1}$ to be

$$\underline{A}^{-1} = \begin{bmatrix} \frac{1}{3} & -\frac{2}{3} & 0 \\ \frac{1}{9} & \frac{7}{9} & -\frac{1}{3} \\ -\frac{1}{9} & \frac{2}{9} & \frac{1}{3} \end{bmatrix}$$

So,

$$\vec{x} = \underline{A}^{-1}\vec{b} = \begin{bmatrix} \frac{1}{3} & -\frac{2}{3} & 0 \\ \frac{1}{9} & \frac{7}{9} & -\frac{1}{3} \\ -\frac{1}{9} & \frac{2}{9} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} 8 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{7}{3} \\ \frac{5}{3} \end{bmatrix}$$

or,

$$x_1 = 0 \qquad x_2 = \frac{7}{3} \qquad x_3 = \frac{5}{3}$$

We presented subroutines in Chapter 3 for the matrix inverse operation and matrix multiplication. When used in succession, they can be employed to solve for the values of $x_1, x_2, \ldots, x_n$. A subroutine to utilize these subroutines is given in Figure 5.1. A main program to read in the set of equations is presented in Figure 5.2.

EXAMPLE 2. Use the main program in Figure 5.2, to solve the following set of simultaneous linear equations.

$$4x_1 + 4x_2 + x_3 = 10$$

$$x_1 + 2x_2 = 8$$

$$x_1 + x_2 + x_3 = 9$$

SOLUTION.

$$x_1 = -7.33 \qquad x_2 = 7.667 \qquad x_3 = 8.67$$

Finally, you should note that this procedure works only if the inverse matrix can be found. If not, then a unique solution does not exist and this method cannot be used.

5.3 Gauss-Jordan Method

Another analytic procedure is the Gauss-Jordan method. This is very closely related to the Gaussian elimination process for finding the inverse of a matrix. We will again use elementary row operations, but the inverse matrix will not be found.

```
18000  REM  SUBROUTINE FOR SOLUTION USING MATRIX INVERSE
18010  REM  THIS SUBROUTINE USES THE MATRIX INVERSE APPROACH
18020  REM  TO SOLVE A SET OF SIMULTANEOUS LINEAR EQUATIONS.
18030  REM  THIS SHOULD BE CALLED FROM A MAIN PROGRAM WHICH
18040  REM  PROVIDES THE COEFFICIENT MATRIX IN THE ARRAY A,
18050  REM  THE RIGHT HAND SIDE VECTOR IN ARRAY B, AND THE
18060  REM  NUMBER OF EQUATIONS IN IE AND NUMBER OF VARIABLES
18070  REM  IN IV (THESE NUMBERS SHOULD BE THE SAME).
18080  DIM A1(IE,IV),B1(IE,1): REM   SET DIMENSIONS FOR USE IN OTHER SUBROUTINES
18090 IROW = IE
18100 ICOL = IV
18110 JROW = IE
18120 JCOL = 1
18130 INV = 1: REM               SET INVERSE FLAG FOR USE IN SUBROUTINES
18140  FOR I = 1 TO IE
18150  FOR J = 1 TO IV
18160 A1(I,J) = A(I,J): REM      SET ARRAY A1 VALUES
18170  NEXT J,I
18180  GOSUB 18850: REM         MAKES A1 UPPER TRIANGULAR
18190  IF DET = 0 THEN  PRINT "EQUATIONS ARE NOT INDEPENDENT"
18200  IF DET = 0 THEN  GOTO 18350
18210  GOSUB 19400: REM         MAKES A1 LOWER TRIANGULAR
18220  FOR I = 1 TO IE
18230 B1(I,1) = B(I): REM        SET ARRAY B1 VALUES
18240  FOR J = 1 TO IV
18250 A1(I,J) = E1(I,J + IV): REM  RESET A1 AS THE INVERSE OF THE A MATRIX
18260  NEXT J,I
18270 ICOL = IV: REM             RESET ICOL TO ITS ORIGINAL VALUE
18280  GOSUB 13750: REM         FROM MATOP; MULTIPLY A1 TIMES B1
18290  REM  AB1 CONTAINS THE SOLUTIONS FOR THE X VALUES
18300  REM  PRINT THE RESULTS
18310  PRINT "RESULTS USING MATRIX INVERSION TECHNIQUE"
18320  FOR I = 1 TO IV
18330  PRINT "X(";I;")= ";AB1(I,1)
18340  NEXT I
18350  RETURN
```

Figure 5.1—Solving Simultaneous Equations, Matrix Inverse Subroutine

The advantage of this method over the inverse method is that if the inverse does not exist, a determination can still be made regarding the status of the solution. Let us expand upon this for a moment. If the inverse of the coefficient matrix cannot be found, then one of two cases holds; either there exists no solution or there is an infinite number of solutions. The existing solution is dependent on the rank of the coefficient matrix and the rank of the matrix created by augmenting the coefficient matrix with the right-hand side vector. The specifics of matrix "rank" and its relationship to the existence of solutions may be found in Hadley (1961), Schmidt (1974), and Stewart (1973). We will not concern ourselves with the theory here, but we will use the concepts in determining the status of the solution.

The Gauss-Jordan method begins by augmenting the coefficient matrix with the right-hand side vector. From the set of equations, we have,

$$\underline{A}\vec{x} = \vec{b}$$

yielding the augmented matrix

$$\underline{AB} = (\underline{A} \mid \vec{b})$$

```
100  REM   MAIN PROGRAM TO READ IN EQUATIONS FOR
110  REM   SIMULTANEOUS SOLUTION.
120  REM   THE NUMBER OF EQUATIONS SHOULD EQUAL THE
130  REM   NUMBER OF VARIABLES
140  INPUT "INPUT THE NUMBER OF VARIABLES ? ";IV
150  INPUT "INPUT THE NUMBER OF EQUATIONS ? ";IE
160  IF IV = IE THEN   GOTO 190
170  PRINT "NUMBER OF EQUATIONS<>NUMBER OF VARIABLES RE-ENTER"
180  GOTO 140
190  DIM B(IE),A(IE,IV)
200  PRINT : PRINT
210  PRINT "FOR EACH EQUATION, INPUT THE FOLLOWING   INFORMATION"
220  PRINT
230  FOR I = 1 TO IE
240  PRINT "FOR EQUATION ";I
250  INPUT "ENTER THE RIGHT HAND SIDE VALUE ? ";B(I)
260  PRINT
270  FOR J = 1 TO IV
280  PRINT "ENTER THE COEFFICIENT OF VARIABLE ";J
290  INPUT A(I,J)
300  NEXT J,I
310  REM   AT THIS TIME, USE APPROPRIATE PROCEDURE TO
320  REM   SOLVE THE SET OF SIMULTANEOUS EQUATIONS
330  GOSUB 18000
340  END
```

Figure 5.2—Matrix Inverse Main Program

Just as in finding the inverse of a matrix, we want to perform row operations necessary to reduce $\underline{A}$ to an identity matrix. If these row operations are performed on the augmented matrix $\underline{AB}$ then they are performed on the vector $\vec{b}$ as well as $\underline{A}$. The result of this procedure yields the solution to the set of equations. The solution is found in the modified $\vec{b}$ vector.

EXAMPLE 3. Find the solution to the following set of linear equations using the Gauss-Jordan method.

$$3x_1 + 2x_2 + 2x_3 = 8$$
$$x_2 + x_3 = 4$$
$$x_1 + 3x_3 = 5$$

SOLUTION. The first step in the solution process is to form the augmented matrix $\underline{AB}$.

$$\underline{AB} = \left[\begin{array}{ccc|c} 3 & 2 & 2 & 8 \\ 0 & 1 & 1 & 4 \\ 1 & 0 & 3 & 5 \end{array}\right]$$

We now want to make the A portion of AB both upper- and lower-triangular. Step by step, we obtain:

$$\underline{AB}_1 = \left[\begin{array}{ccc|c} 3 & 2 & 2 & 8 \\ 0 & 1 & 1 & 4 \\ 0 & -\frac{2}{3} & \frac{7}{3} & \frac{7}{3} \end{array}\right] \qquad \underline{AB}_2 = \left[\begin{array}{ccc|c} 3 & 2 & 2 & 8 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 3 & 5 \end{array}\right]$$

$$\underline{AB}_3 = \left[\begin{array}{ccc|c} 3 & 2 & 2 & 8 \\ 0 & 1 & 0 & \frac{7}{3} \\ 0 & 0 & 3 & 5 \end{array}\right] \qquad \underline{AB}_4 = \left[\begin{array}{ccc|c} 3 & 2 & 0 & \frac{14}{3} \\ 0 & 1 & 0 & \frac{7}{3} \\ 0 & 0 & 3 & 5 \end{array}\right]$$

$$\underline{AB}_5 = \left[\begin{array}{ccc|c} 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{7}{3} \\ 0 & 0 & 3 & 5 \end{array}\right]$$

At this point we have effectively reduced the original equations to the following set of easily solvable equations:

$$3x_1 = 0$$

$$x_2 = \tfrac{7}{3}$$

$$3x_3 = 5$$

The solution is easily seen to be:

$$x_1 = 0 \qquad x_2 = \tfrac{7}{3} \qquad x_3 = \tfrac{5}{3}$$

Note that these values would be found in the revised augmented matrix if the $\underline{A}$ portion were reduced to an identity matrix. Thus,

$$\underline{AB}_6 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & \frac{7}{3} \\ 0 & 0 & 1 & \frac{5}{3} \end{array}\right]$$

You can see that this is the same solution that was found in Example 1.

The value of this approach is evident if the inverse of A does not exist. In this case, either the equations are "inconsistent," yielding no solution, or one or more equations are "redundant," yielding an infinite number of solutions. We are able to determine which of these conditions holds using the Gauss-Jordan method.

If the inverse of $\underline{A}$ does not exist, then its determinant is zero. Using the Gauss-Jordan method, after all row operations have been completed, at least one row will exist in the A portion of $\underline{AB}$ which has only zeros. This row portrays no information. The corresponding row in the $\vec{b}$ portion of the revised $\underline{AB}$ matrix is of critical importance. If it

is also zero, then no information is available from the entire row, and thus a redundant equation is represented. If it is not zero, the resulting relationship implies that zero is equal to a constant different from zero, which clearly cannot be true. In this case, the equation is inconsistent, and there exists no solution that simultaneously satisfies all equations.

This may be easier to see in an example. Example 4 below illustrates the determination process. You should notice that both conditions exist in this problem. Of course, the overriding factor is inconsistency. It need only occur once to result in a conclusion of no solution.

EXAMPLE 4. Determine whether the following set of simultaneous linear equations has a unique solution, no solution, or an infinite number of solutions.

$$2x_1 + 2x_2 + 4x_3 = 12$$

$$4x_1 + 4x_2 + 8x_3 = 20$$

$$x_1 + x_2 + 2x_3 = 6$$

SOLUTION.

$$\underline{AB}_1 = \left[\begin{array}{ccc|c} 2 & 2 & 4 & 12 \\ 4 & 4 & 8 & 20 \\ 1 & 1 & 2 & 6 \end{array}\right] \qquad \underline{AB}_2 = \left[\begin{array}{ccc|c} 2 & 2 & 4 & 12 \\ 0 & 0 & 0 & -4 \\ 1 & 1 & 2 & 6 \end{array}\right]$$

$$\underline{AB}_3 = \left[\begin{array}{ccc|c} 2 & 2 & 4 & 12 \\ 0 & 0 & 0 & -4 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

equation representation:

$$2x_1 + 2x_2 + 4x_3 = 12$$

$$0 = -4 \text{ (inconsistent)}$$

$$0 = 0 \text{ (redundant)}$$

CONCLUSION. Due to inconsistent equation, no solution exists.

You probably noticed another interesting development in Example 4. When the inverse of $\underline{A}$ does not exist, you cannot reduce $\underline{A}$ to an identity. Thus, the solutions are not evident from the revised $\underline{AB}$ matrix. This makes sense, since there is no solution for inconsistent equations, and it is impossible to present an infinite number of solutions.

If the determination is an infinite number of solutions, this means that a certain number of variables may be set to any arbitrary values and the equations solved for the remaining variables. If there are n equations and r of the equations are redundant, then r of the n variables may be assigned arbitrary values barring inconsistent equations. However, redundant equations will be pinpointed.

Figure 5.3 presents the subroutine for the Gauss-Jordan method. It utilizes row operations to achieve both upper- and lower-triangularization. If a solution does not exist for your set of linear equations, the program will tell you. If there are an infinite number of solutions, this will also be specified. Note that revised versions of the upper-triangular and lower-triangular subroutines from Chapter 3 are included in the Gauss-Jordan procedure in Figure 5.3. Revisions were necessary to pinpoint inconsistent or redundant equations.

EXAMPLE 5. Use the subroutine presented for the Gauss-Jordan subroutine to solve the following set of simultaneous equations.

$$\begin{aligned} 3x_1 + 8x_2 - 4x_3 + x_4 &= 35 \\ 2x_2 + x_3 - 3x_4 &= 27 \\ 11x_1 + 5x_4 &= 20 \\ x_3 + x_4 &= 9 \end{aligned}$$

SOLUTION. Using this program and the main program in Figure 5.2, the final solution is found to be:

$$x_1 = 1.9676 \quad x_2 = 8.3426 \quad x_3 = 9.3287 \quad x_4 = -0.3287$$

5.4 Gauss-Seidel Method

The Gauss-Seidel method is a numerical procedure for solving simultaneous linear equations. It is iterative in nature. Because it is a numerical approach, there is no guarantee that the solution will be found.

To explain the procedure, let us look at the general form again.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \quad\quad\quad \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

Suppose we reorder these equations so that the coefficient of x in the jth equation, a_{ij}, is larger in magnitude than the remaining coefficients in that equation. We can rewrite the equations so that each represents the solution of one variable, as follows:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n) \\ x_2 &= \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - a_{24}x_4 - \cdots - a_{2n}x_n) \\ &\vdots \\ x_n &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{nn-1}x_{n-1}) \end{aligned} \tag{1}$$

```
18360  REM  ****************************************************
18370  REM  GAUSS-JORDAN SUBROUTINE
18380  REM  THIS SUBROUTINE USES THE GAUSS-JORDAN PROCEDURE TO SOLVE SETS
18390  REM  OF SIMULTANEOUS LINEAR EQUATIONS.IT USES ADDITIONAL SUBROUTINES
18400  REM  WHICH WERE PRESENTED IN CHAPTER 3 AND ARE CONTAINED IN THE FILE
18410  REM  MATOP.  THIS FILE MUST BE MERGED PRIOR TO SOLUTION USING THIS
18420  REM  SUBROUTINE. OTHER INPUTS ARE THE NUMBER OF EQUATIONS (IE),THE
18430  REM  NUMBER OF VARIABLES (IV),THE VARIABLE COEFFICIENTS IN ARRAY A,
18440  REM  AND THE RIGHT HAND SIDE VECTOR STORED  IN ONE-DIMENSIONAL ARRAY B.
18450  DIM A1(IE,IV),B1(IE,1),D%(IV): REM   DIMENSION REQUIRED ARRAYS
18460 IROW = IE
18470 ICOL = IV
18480 JROW = IE
18490 JCOL = 1
18500  FOR I = 1 TO IE
18510  FOR J = 1 TO IV
18520 A1(I,J) = A(I,J): REM          SET VALUES FOR MATRIX A1
18530  NEXT J
18540 B1(I,1) = B(I): REM            SET VALUES FOR MATRIX B1
18550 D%(I) = 1
18560  NEXT I
18570 INV = 2: REM                   SET INVERSE FLAG FOR UPCOMING SUB.
18580  GOSUB 18850: REM             REVISED UPPER TRIANGULAR ROUTINE
18590  GOSUB 19400: REM             REVISED LOWER TRIANGULAR ROUTINE
18600  REM  THE DESIRED RESULT IS IN THE LAST COLUMN OF E1
18610  PRINT : PRINT
18620  PRINT : PRINT
18630  PRINT "RESULTS FROM GAUSS-JORDAN REDUCTION"
18640  REM  FIRST, CHECK FOR INCONSISTENT OR REDUNDANT EQUATIONS.
18650 ST% = 0: REM                   SET UNIQUE SOLUTION FLAG
18660  FOR I = 1 TO IROW
18670  IF D%(I) <  > 0 THEN  GOTO 18750
18680 ST% = 1: REM                   SET REDUNDANT EQUATIONS FLAG
18690  FOR J = IROW + 1 TO ICOL
18700  IF E1(I,J) = 0 THEN  GOTO 18730
18710 ST% = 2: REM                   SET INCONSISTENT EQUATIONS FLAG
18720  GOTO 18790
18730  NEXT J
18740  REM  NORMAL EXIT MEANS REDUNDANT EQUATION
18750  NEXT I
18760  IF ST% = 0 THEN  GOTO 18810: REM  UNIQUE SOLUTION
18770  PRINT "THIS SET OF EQUATIONS HAS AN INFINITE # OF SOLUTIONS"
18780  GOTO 18840
18790  PRINT "THIS SET OF EQUATIONS IS INCONSISTENT;NO SOLUTION"
18800  GOTO 18840
18810  FOR I = 1 TO IV
18820  PRINT "X(";I;") = ";E1(I,IV + 1)
18830  NEXT I
18840  RETURN
18850  REM  ****************************************************
18860  REM  UPPER TRIANGULAR SUBROUTINE (REVISED) :
18870  REM  SUBROUTINE TO CREATE AN UPPER TRIANGULAR MATRIX
18880  REM  PROVIDE MATRIX A1 AND ITS DIMENSIONS,IROW=ICOL.RESULT IN E1
18890  IF IROW <  > ICOL THEN  PRINT "NUMBER OF ROWS<>NUMBER OF COLUMNS:ERROR"
18900  IF IROW = ICOL THEN  GOSUB 18920
18910  RETURN
18920  REM  ****************************************************
18930  REM  UPPER TRI ROUTINE (REVISED)
18940  REM    UPPER TRIANGULAR ROUTINE
18950  REM     ACCESS ONLY THROUGH SUB 15810
18960  IF INV = 1 THEN ICOL = 2 * ICOL: REM     SET PARAMETER FOR MATRIX INVERSE
18970  IF INV <  > 1 THEN ICOL = ICOL + 1: REM    SET PARAMETER FOR GAUSS-JORDAN
18980  DIM E1(IROW,ICOL)
18990  FOR I = 1 TO IROW
19000  FOR J = 1 TO IROW
19010 E1(I,J) = A1(I,J): REM        ASSIGN E1 MATRIX TO A1
```

```
19020  IF INV < > 1 THEN  GOTO 19050: REM     SKIP IDENTITY AUGMENTATION
19030  REM  THIS SECTION IS USED ONLY IF A MATRIX IS BEING INVERTED
19040  IF I = J THEN E1(I,J + IROW) = 1: REM    AUGMENT THE IDENTITY MATRIX TO E1
19050  NEXT J
19060  IF INV < > 1 THEN F1(I,IROW + 1) = B1(I,1): REM  FOR THE GAUSS JORDAN PROCEDURE
19070  NEXT I
19080 DET = 1: REM                          SET DETERMINANT FLAG
19090  FOR I = 1 TO IROW
19100  IF I < > IROW THEN  GOTO 19140
19110  IF E1(I,I) < > 0 THEN  GOTO 19240
19120 DT = 0
19130  GOTO 19160
19140  IF E1(I,I) < > 0 THEN  GOTO 19190
19150  GOSUB 19280: REM         DIAGONAL ELEMENT MUST NOT BE ZERO
19160  IF DT = 0 THEN DET = 0
19170  IF DT = 0 THEN D%(I) = 0
19180  IF DT = 0 THEN  GOTO 19240
19190  FOR J = I + 1 TO IROW
19200 XM = E1(J,I) / E1(I,I): REM  MULTIPLIER TO ZERO THE COLUMN ELEMENTS
19210  FOR K = I TO ICOL
19220 E1(J,K) = E1(J,K) - XM * E1(I,K): REM    CALCULATE NEW ELEMENTS
19230  NEXT K,J
19240  NEXT I
19250  RETURN
19260  REM  ******************************************************
19270  REM  ZERO DETERMINANT DETECTION SUBROUTINE (REVISED):
19280  REM   SUBROUTINE TO ENSURE DIAGONAL ELEMENTS ARE NONZERO DURING
19290  REM   UPPER-TRIANGULARIZATION.IF THIS IS NOT POSSIBLE,DET IS ZERO
19300 DT = 1
19310  FOR J = I + 1 TO IROW
19320  IF E1(J,I) = 0 THEN  GOTO 19370
19330  FOR K = 1 TO ICOL
19340 E1(I,K) = E1(I,K) + E1(J,K): REM       ADD ROWS TO MAKE DIAGONAL NONZERO
19350  NEXT K
19360  RETURN
19370  NEXT J
19380 DT = 0: REM                           DET MUST BE ZERO AT THIS POINT
19390  RETURN
19400  REM  ******************************************************
19410  REM  LOWER TRIANGULARIZATION SUBROUTINE (REVISED):
19420  REM  SUBROUTINE TO MAKE A MATRIX LOWER TRIANGULAR,THEN AN IDENTITY.
19430  REM  FIND THE MULTIPLIER TO ZERO ELEMENTS IN THE KTH COLUMN
19440  REM  ABOVE THE DIAGONAL.
19450  REM  INPUT IS THE MATRIX E1 WITH ITS DIMENSIONS IROW AND ICOL
19460  FOR IJ = 1 TO IROW
19470  IF IJ = IROW THEN  GOTO 19560
19480 IK = IROW - IJ + 1
19490  FOR I = 1 TO IK - 1
19500  IF INV = 1 THEN  GOTO 19520
19510  IF D%(IK) = 0 THEN  GOTO 19560
19520 XM = E1(I,IK) / E1(IK,IK)
19530  FOR J = I + 1 TO ICOL
19540 E1(I,J) = E1(I,J) - XM * E1(IK,J)
19550  NEXT J,I
19560  NEXT IJ
19570  IF DET = 0 THEN  GOTO 19650
19580  REM  CREATE IDENTITY BY MULTIPLYING EACH ROW BY THE RECIPROCAL OF THE
19590  REM  DIAGONAL ELEMENT OF THE REVISED MATRIX
19600  FOR I = 1 TO IROW
19610 DIV = E1(I,I)
19620  FOR J = 1 TO ICOL
19630 E1(I,J) = E1(I,J) / DIV
19640  NEXT J,I
19650  RETURN
```

Figure 5.3—Gauss-Jordan Subroutine

We can develop initial estimates for each variable in turn. First, approximate x_1 by

$$x_1^* = \frac{b_1}{a_{11}}$$

Remaining initial approximations are found as:

$$x_2^* = \frac{1}{a_{22}}(b_2 - a_{21}x_1^*)$$

$$x_3^* = \frac{1}{a_{33}}(b_3 - a_{31}x_1^* - a_{32}x_2^*)$$

$$\vdots$$

$$x_n^* = \frac{1}{a_{nn}}(b_n - a_{n1}x_1^* - a_{n2}x_2^* - \cdots - a_{nn-1}x_{n-1}^*)$$

Based upon these initial approximations, the set of equations (1) is solved for each variable. These solutions are then used as inputs to re-solve equations (1), and the iterative process continues. One important aspect of the Gauss-Seidel method, that distinguishes it from the similar Jacobi method (Hildebrand, 1974) is that in solving for each variable during each iteration, the most recently calculated values of the input variables are used. For example, when you find x_2, the input value for x_1 is the value just calculated; not the one found during the previous iteration. If we specify x_j as the value calculated during the previous iteration, and x_j^* as the new value, the iterative calculations are performed as follows:

$$x_1^* = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n)$$

$$x_2^* = \frac{1}{a_{22}}(b_2 - a_{21}x_1^* - a_{23}x_3 - \cdots - a_{2n}x_n)$$

$$x_3^* = \frac{1}{a_{33}}(b_3 - a_{31}x_1^* - a_{32}x_2^* - a_{34}x_4 - \cdots - a_{3n}x_n) \qquad (2)$$

$$\vdots$$

$$x_n^* = \frac{1}{a_{nn}}(b_n - a_{n1}x_1^* - a_{n2}x_2^* - \cdots - a_{nn-1}x_{n-1}^*)$$

The iterative process continues until successive solutions are within a certain tolerance. Remember, however, that the procedure may not converge. It is therefore wise to limit the number of iterations or provide a check for divergence.

The Gauss-Seidel method is too lengthy to solve large problems by hand. However, we illustrate the procedure on a small problem in the following example.

EXAMPLE 6. Solve the following set of simultaneous linear equations using the Gauss-Seidel method.

$$\begin{aligned} x_1 + 8x_2 + 2x_3 &= 4 \\ 9x_1 + x_2 - x_3 &= 15 \\ 2x_2 + 6x_3 &= 9 \end{aligned}$$

SOLUTION. The first step in the solution is to arrange the equations so that the large values are on the diagonal of the coefficient matrix.

$$\begin{aligned} 9x_1 + x_2 - x_3 &= 15 \\ x_1 + 8x_2 + 2x_3 &= 4 \\ 2x_2 + 6x_3 &= 9 \end{aligned}$$

We now find the initial estimates for each variable.

$$x_1^* = \frac{b_1}{a_{11}} = \frac{15}{9} = 1.6667$$

$$x_2^* = \frac{1}{a_{22}}(b_2 - a_{21}x_1^*) = \frac{1}{8}(4 - 1.6667) = \frac{7}{24} = 0.2917$$

$$x_3^* = \frac{1}{a_{33}}(b_3 - a_{31}x_1^* - a_{32}x_2^*) = \frac{1}{6}\left(9 - 0 - \frac{7}{12}\right) = \frac{101}{72} = 1.4028$$

Using the iterative formulas (2), the following table is produced. Compare the values with those you found.

Iteration	x_1	x_2	x_3
0	1.6667		
1	1.7647	−.0140	1.5047
2	1.8354	−.1056	1.5352
3	1.8490	−.1149	1.5383
4	1.8504	−.1159	1.5386
5	1.8505	−.1160	1.5387

You should find it useful to examine conditions under which the Gauss-Seidel method is guaranteed to converge. Hildebrand (1974) presents these conditions as:

1. The coefficient matrix must not contain a submatrix with p rows and q columns whose elements are all zero, where $p+q=n$.
2. The magnitude of the diagonal elements of the coefficient matrix must be at least the sum of the magnitudes of the other elements in that row (equation), and is larger than the sum in at least one case.

If these conditions hold, the method will converge. However, the method may still converge even if the conditions do not hold. Thus, the conditions are sufficient but not necessary for convergence.

The convergence criterion we will use is based on comparison of successive points. If all points fall within a specified amount of their previous value, the process is terminated. Divergence is handled through a limit on the number of iterations. If 100 iterations are too restrictive, you may want to establish your own iteration limit.

The Gauss-Seidel subroutine is presented in Figure 5.4. The main program from Figure 5.2 can be used to access it. This program is used in the following example.

EXAMPLE 7. Use the main program in Figure 5.2 to find the solution to the following set of linear equations.

$$x_1 + 14x_2 + 3x_3 + 6x_4 = 25$$

$$10x_1 + 2x_2 - x_3 - 2x_4 = 18$$

$$-x_1 - 4x_2 + 12x_3 + 2x_4 = 31$$

$$2x_1 + 3x_2 + x_3 + 8x_4 = 14$$

SOLUTION. From the computer output, the results are:

$x_1 = 2.055$ $x_2 = 0.764$ $x_3 = 2.912$ $x_4 = 0.586$

```
19660  REM  ******************************************************
19670  REM  GAUSS-SEIDEL SUBROUTINE
19680  REM  THIS SUBROUTINE USES THE GAUSS-SEIDEL PROCEDURE TO SOLVE SETS
19690  REM  OF SIMULTANEOUS LINEAR EQUATIONS.INPUTS PROVIDED TO THIS
19700  REM  SUBROUTINE MUST BE THE FOLLOWING:
19710  REM     1. A SET OF LINEAR EQUATIONS.THE VARIABLE COEFFICIENTS
19720  REM        MUST BE STORED IN THE TWO-DIMENSIONAL ARRAY A AND THE
19730  REM        RIGHT HAND SIDE VALUES MUST BE STORED IN THE
19740  REM        ONE-DIMENSIONAL ARRAY B.
19750  REM     2. DIMENSIONS MUST BE PROVIDED IN IE (NUMBER OF EQUATIONS)
19760  REM        AND IV (NUMBER OF VARIABLES).
19770  REM     3. THE NUMBER OF ROWS ARE ASSUMED TO BE EQUAL TO THE
19780  REM        NUMBER OF VARIABLES. THAT IS, IE=IV.
19790  REM  THE USER MUST INPUT TO THIS SUBROUTINE THE G-S CONVERGENCE
19800  REM  CRITERION, E.
19810  DIM ID(IE)
19820  INPUT "SPECIFY THE CONVERGENCE CRITERION TO BE USED ? ";E
19830  PRINT : PRINT
19840  INPUT "WOULD YOU LIKE TO SEE RESULTS FROM ALL  ITERATIONS (Y/N) ? ";T$
19850  REM  FIND THE LARGEST COEFFICIENT IN EACH EQUATION
19860  FOR I = 1 TO IE
19870  FOR J = 1 TO IV
19880  IF J <  > 1 THEN  GOTO 19920
19890 MAX =  ABS (A(I,J))
19900 ID(I) = J: REM              SET LARGEST VALUE COLUMN
19910  GOTO 19950
19920  IF  ABS (A(I,J)) < MAX THEN  GOTO 19950
```

```
19930 MAX =  ABS (A(I,J))
19940 ID(I) = J: REM              SET LARGEST VALUE COLUMN
19950  NEXT J,I
19960  REM  SWITCH ROWS TO PUT THE LARGEST VALUES ON THE DIAGONALS
19970  FOR I = 1 TO IE - 1
19980  IF ID(I) = I THEN  GOTO 20130
19990  FOR II = I TO IE
20000  IF ID(II) <  > I THEN  GOTO 20030
20010 JJ = II
20020  GOTO 20040
20030  NEXT II
20040  FOR J = 1 TO IV
20050 D = A(JJ,J)
20060 A(JJ,J) = A(I,J): REM            SWITCH ROWS FOR LARGEST ON DIAGONAL
20070 A(I,J) = D
20080  NEXT J: REM                    RESET LARGEST VALUE COLUMN
20090 D = B(I)
20100 B(I) = B(JJ): REM                SWITCH RIGHT HAND SIDE VALUES
20110 B(JJ) = D
20120 ID(JJ) = ID(I): REM              RESET LARGEST VALUE COLUMN
20130  NEXT I
20140  REM  NOW PERFORM THE CALCULATIONS NECESSARY FOR THE
20150  REM   GAUSS-SEIDEL PROCEDURE.
20160  REM
20170  REM  FIND INITIAL APPROXIMATIONS
20180  FOR J = 1 TO IE
20190 SUM = 0
20200  IF J = 1 THEN  GOTO 20240
20210  FOR I = 1 TO J - 1
20220 SUM = SUM + X(I)
20230  NEXT I
20240 X(J) = (B(J) - SUM) / A(J,J)
20250  NEXT J
20260  REM  SOLVE VIA THE ITERATIVE TECHNIQUE
20270 IK = 0: REM             SET ITERATION COUNTER
20280 K% = 1: REM             SET CONVERGENCE FLAG
20290 IK = IK + 1: REM          UPDATE ITERATION COUNTER
20300  IF IK > 100 THEN  GOTO 20560: REM     NO CONVERGENCE AFTER 100 ITERATIONS
20310  FOR I = 1 TO IE
20320 SUM = 0
20330  FOR J = 1 TO IV
20340  IF I = J THEN  GOTO 20360
20350 SUM = SUM + A(I,J) * X(J)
20360  NEXT J
20370 XX = X(I)
20380 X(I) = (B(I) - SUM) / A(I,I)
20390  IF  ABS (XX - X(I)) > E THEN K% = 2: REM   NOT YET CONVERGED
20400  NEXT I
20410  IF T$ = "N" THEN  GOTO 20480
20420  PRINT : PRINT : PRINT
20430  PRINT "FOR ITERATION NUMBER ";IK
20440  FOR I = 1 TO IV
20450  PRINT "X(";I;") = ";X(I)
20460  NEXT I
20470  INPUT "ENTER CARRIAGE RETURN TO CONTINUE -> ";F$
20480  IF K% = 2 THEN  GOTO 20280
20490  REM  CONVERGENCE HAS OCCURRED; PRINT RESULTS
20500  PRINT "THE GAUSS-SEIDEL PROCEDURE HAS CONVERGED"
20510  PRINT "TO THE FOLLOWING SOLUTION"
20520  FOR I = 1 TO IV
20530  PRINT "X(";I;") = ";X(I)
20540  NEXT I
20550  GOTO 20570
20560  PRINT "NO CONVERGENCE AFTER 100 ITERATIONS :   PROCESS TERMINATING"
20570  RETURN
```

Figure 5.4—Gauss-Seidel Subroutine

5.5 Summary

We have presented three methods for solving simultaneous linear equations. Each has advantages and disadvantages; the particular situation will dictate which one to use. For small problems, the matrix inverse or Gauss-Jordan procedures may be preferable. However, for a large number of equations, the Gauss-Seidel method should be considered due to the number of calculations involved, provided that the conditions are met for its use. The solution to simultaneous linear equations is an important tool. We will see in Chapter 9 how they are used in solving linear programming problems.

References

Hadley, G., *Linear Algebra*. Addison-Wesley Publishing Company, Inc., Reading, Mass., 1961.

Hildebrand, F. B., *Introduction to Numerical Analysis*. McGraw-Hill Book Co., New York, N.Y., 1974.

Schmidt, J. William, *Mathematical Foundations for Management Science and Systems Analysis*. Academic Press, Inc., New York, N.Y., 1974.

Schmidt, J. William and Robert P. Davis, *Foundations of Analysis in Operations Research*. Academic Press, Inc., New York, N.Y., 1981.

Stewart, G. W., *Introduction to Matrix Computations*. Academic Press, Inc., New York, N.Y., 1973.

Exercises

1. The following set of simultaneous linear equations has an infinite number of solutions.

$$\begin{aligned} 6x_1 \quad &+ 3x_3 + x_4 - 2x_5 = 7 \\ 5x_1 - x_2 &+ 2x_3 + 2x_4 \quad = 6 \\ 3x_2 - x_3 &+ 5x_4 + 4x_5 = 8 \end{aligned}$$

   Write a computer program which presents numerous solutions by setting two variables at a time equal to arbitrary values and solving for the remaining three variables.

2. Specify whether the following sets of equations have a unique solution, no solution, or an infinite number of solutions.

   a. $\begin{aligned} 3x_1 + 2x_2 &= 5 \\ x_1 - x_2 &= 7 \\ 4x_1 \quad &= 12 \end{aligned}$

   b. $\begin{aligned} 2x_1 + 2x_2 + x_3 - x_4 &= 7 \\ x_1 - 3x_2 + 2x_3 - 2x_4 &= 5 \\ 3x_1 + x_2 - x_3 + x_4 \quad &= 6 \end{aligned}$

   c. $\begin{aligned} 3x_1 + x_2 &= 6 \\ 9x_1 + 3x_2 &= 15 \end{aligned}$

   d. $\begin{aligned} 8x_1 + 7x_2 - 3x_3 &= 8 \\ 4x_1 - x_2 + 6x_3 &= 12 \end{aligned}$

3. Modify the computer subroutines presented in this chapter to calculate the number of mathematical operations involved in solving a set of simultaneous linear equations. Compare the number of operations required for each technique as the number of equations and variables increases (base your comparisons on the same set of equations). What conclusions can you draw from this exercise?

4. For a certain chemical process we have three inputs and three desired output targets. Each ounce of input A yields three units of output 1, one unit of output 2, and one unit of output 3. Each ounce of input B yields one unit of output 1, four units of output 2, and no units of output 3.

Finally, each ounce of input C yields one unit of output 1, two units of output 2, and two units of output 3. Our requirements are that the process results in exactly 12 units of output 1, 16 units of output 2, and 11 units of output 3. How many ounces of each type of input do we need?

5. A "sparse" matrix is one that has a large number of zero elements. What impact does a "sparse" coefficient matrix have on the solution to simultaneous linear equations? Would changes in the programs presented in this chapter be required to realize this impact?

6

Roots of Polynomials

In the course of problem solving, you may often encounter the need to find the root of a function. That is, you want to find the value of the variable that makes the function equate to zero. In simple cases, this can be accomplished easily by hand. Formulas are readily available to find the roots of polynomials of degree three or less. For polynomials of higher degree, if they cannot be factored, the procedure becomes one of trial and error unless numerical techniques are used, in general requiring a computer. This chapter presents a numerical procedure for finding the roots of a polynomial.

6.1 Roots of Polynomials

The root of a function is that value of the variable which equates the function to zero. A function may have many roots, such as sin x. However, in our discussion, we will limit outselves to functions which are polynomials. For simple functions, you may be able to solve for the roots by inspection. Consider the following:

$$f(x) = x^2 - 3$$

You can see that the roots of f(x) are $\sqrt{3}$ and $-\sqrt{3}$. For more complex functions, factoring may be required:

$$f(x) = x^2 - 2x - 3 = (x - 3)(x + 1)$$

Roots: $x = 3, x = -1$

If factoring is not possible, all is not lost. For polynomials of degree two or three there exist formulas for the roots. Following standard forms, these are:

Quadratic

$$f(x) = ax^2 + bx + c$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Cubic

$$f(y) = y^3 + py^2 + qy + r$$

let

$$a = \frac{1}{3}(3q - p^2) \quad b = \frac{1}{27}(2p^3 - 9pq + 27r)$$

$$A = \sqrt[3]{-\frac{b}{2} + \sqrt{\frac{b^2}{4} + \frac{a^3}{27}}} \qquad B = \sqrt[3]{-\frac{b}{2} - \sqrt{\frac{b^2}{4} + \frac{a^3}{27}}}$$

then

$$X = A+B, \; -\frac{A+B}{2} + \frac{A-B}{2}\sqrt{-3}, \; -\frac{A+B}{2} - \frac{A-B}{2}\sqrt{-3}$$

You can see that this gets quite involved.

Notice that there are two roots for a quadratic function and three for a cubic function. This relationship holds in general. An *nth* order polynomial will have n roots, although they may not all be unique. That is, more than one root may have the same value. Also, roots may not be "real." In this case, they are called "complex" and have both a real part and an imaginary part. The imaginary part of a complex number results from the square root of a negative number. In the formulas given above, both real and complex roots are possible. In the numerical technique presented, real roots will be found, with an option to find complex roots.

6.2 Newton-Raphson Method

The Newton-Raphson Method for finding the roots of polynomials is a particular technique of a class of iterative procedures used with nonlinear equations. We will omit discussion of the theoretical development of the method, but will illustrate its intuitive appeal and operational characteristics. For further details, you may want to consult Hildebrand (1974).

General Concepts

Let us begin by examining the way in which the Newton-Raphson method locates a root. It is an iterative method that converges, hopefully, to a root. There are cases in which there is no convergence. We will discuss these later. The general procedure is to specify an initial starting point, x_0, and generate successive points until the root is approximated. Each successive point is found as the point where the tangent line of the function, evaluated at the previous point, intersects the x-axis. By examining Figure 6.1 you can see how this method converges to the root, r.

What about multiple roots? It is evident that the same initial starting point will result in the same root. Therefore, if multiple roots exist you must start in more than one place in order to find them. In fact, there is no guarantee that you will find all the roots. A large number of starting points may lead to the same root. This is a drawback that is not unique to Newton-Raphson. In order to reduce the likelihood of missing a root, a large number of initial starting values are used. This is illustrated for two roots in Figure 6.2.

In examining Figures 6.1 and 6.2 you may have noticed a condition under which the method does not converge to a root or may take an extremely long time to locate a root.

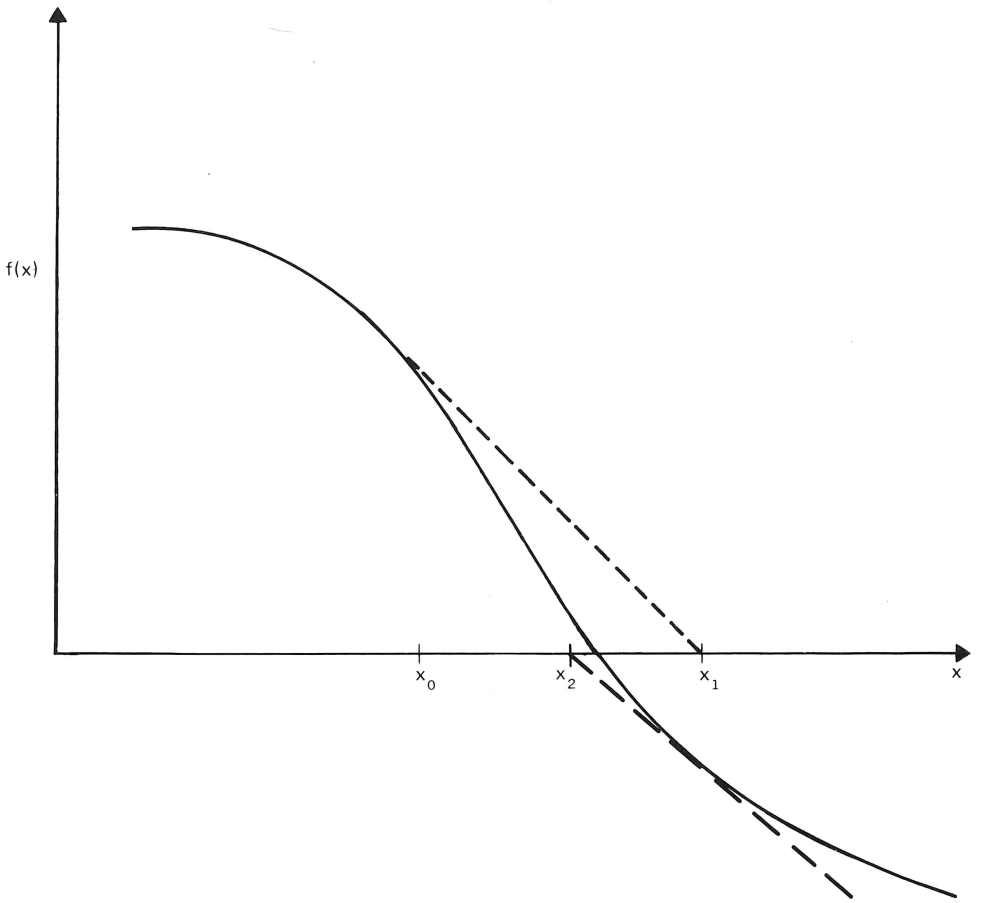


Figure 6.1—The Newton-Raphson Procedure

This unfortunate occurrence results from a point where the slope of the function is at or near zero. In this case either the tangent line never intersects the x-axis (zero slope) or the next point found is extremely far away (slope near zero). We illustrate this situation in Figure 6.3. Fortunately, we can plan ahead to avoid this situation. If, upon calcualtion of the slope at the current point, its value is at or near zero, we can move a particular distance from that point and try again. For many functions this is an effective procedure.

You may run into some polynomials for which this procedure simply will not work. This will be due to the shape of the function and the precise location of the root. However, in general, it is a very reliable procedure.

Operation Requirements

You know what Newton-Raphson does. We will now illustrate how it works. The graphic illustrations presented up to this point cannot be used; if we could graph the function we would know what the roots were. The procedure used is algebraic, and uses iterative relationships to generate each succesive point.

The general formula used is

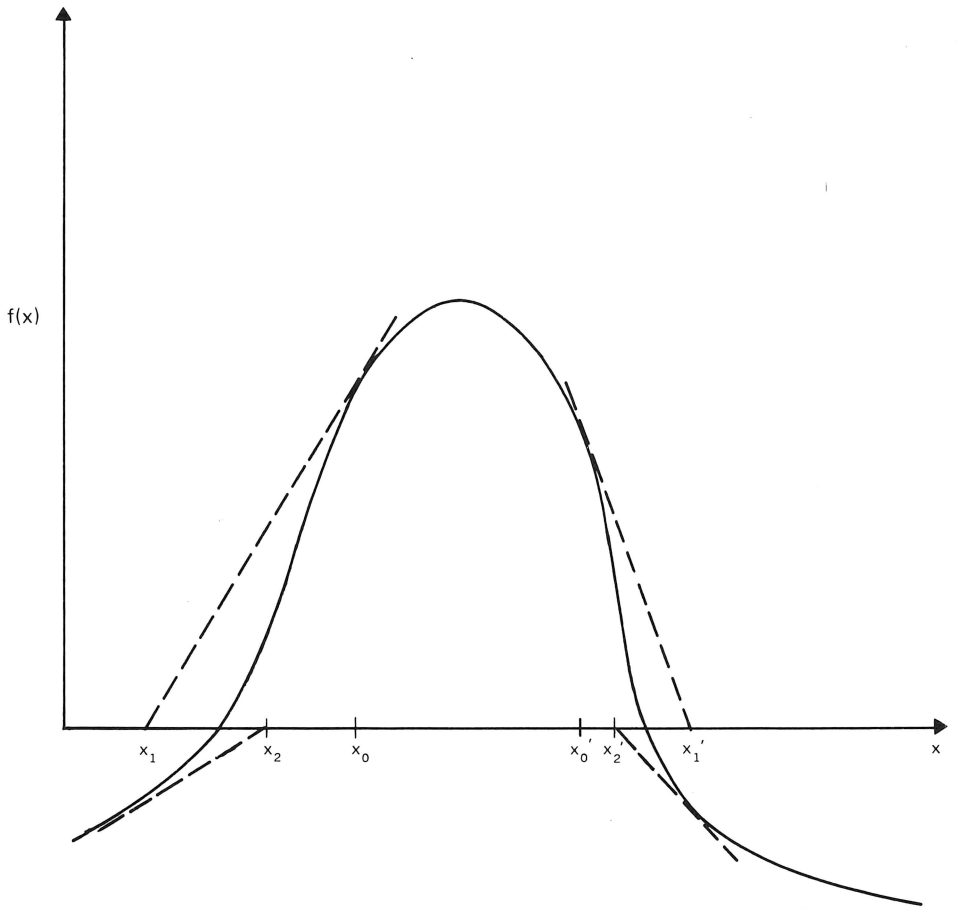


Figure 6.2—Multiple Starting Points and Roots

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{1}$$

where,

k = index of points
$f'(x_k)$ = derivative of the function f(x) evaluated at the point x_k (slope of the tangent line at that point)

You can see the computational problems inherent in the situation depicted in Figure 6.3. A horizontal tangent line has a slope of zero, causing the term $f(x_k)/f'(x_k)$ to be undefined. A slope near zero is almost as bad.

Does this formula make sense to you? Consider the point-slope form of an equation. The tangent line which passes through the point $f(x_k)$ has the general formula

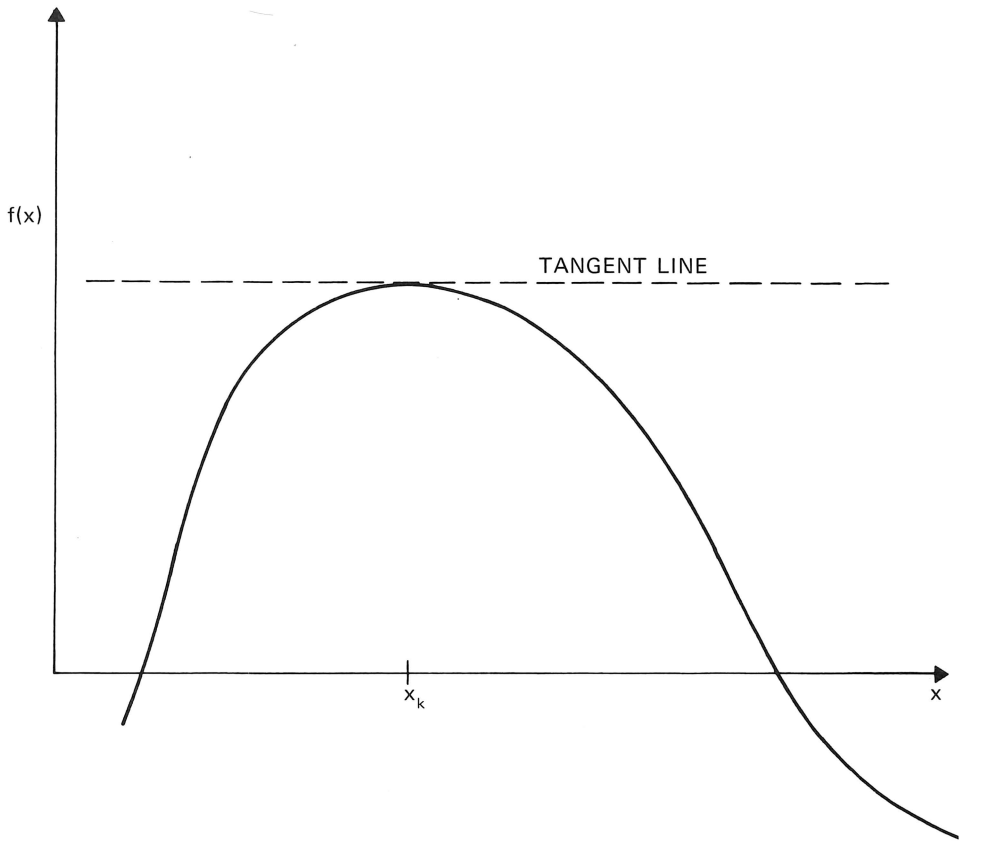


Figure 6.3—Unfortunate Iterative Point

$f(x_k) = ax_k + b$

where,

a = slope of the line

b = intercept with the f(x) axis

Now we know the slope of the line, it is simply $f'(x_k)$. Therefore, we know that

$$f(x_k) = x_k f'(x_k) + b \tag{2}$$

Furthermore, x_{k+1} must lie on the same line. But, at x_{k+1} the function is zero. So,

$f(x_{k+1}) = x_{k+1} f'(x_k) + b$

becomes

$$0 = x_{k+1} f'(x_k) + b \tag{3}$$

To find x_{k+1}, let us subtract (2) from (3) to get

$$x_{k+1}f'(x_k) - x_kf'(x_k) + f(x_k) = 0$$

By collecting terms and solving for x_{k+1}, we get

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

which is equation (1). Therefore, this formula does indeed generate the succession of points desired in the procedure.

You have noticed that the derivative of the function is required for Newton-Raphson. Since we are limiting ourselves to polynomials you should have no difficulty with this. It is automatically included in the computer program.

The remaining issue is determining when to stop. In other words, which x is the root of the polynomial? Of course, if $f(x_k)$ is zero, then $x_{k+1}=x_k$; this can be used as a stopping criterion. In most cases the root will not be exact. A stopping rule based on the absolute value of $f(x_k)$ is approrpriate in these instances.

EXAMPLE 1. Perform the Newton-Raphson method on the following polynomial.

$$f(x) = 7x^4 - 3x^3 + x^2 + x - 8$$

Stop when $| f(x_k) | \leq 0.001$. Start at the point $x_0=3$.

SOLUTION. The table below illustrates the succesion of points in the solution.

k	x_k	$f(x_k)$	$f'(x_k)$	x_{k+1}
0	3	490	682	2.2815
1	2.2815	153.5286	291.2470	1.7544
2	1.7544	46.9423	127.9952	1.3876
3	1.3876	13.2519	61.2616	1.1713
4	1.1713	2.8974	35.9883	1.0908
5	1.0908	0.2969	28.8130	1.0805
6	1.0805	0.0043	27.9737	1.0803
7	1.0803	0.0000		

Computer Program

The example problem above was quite tedious by hand. But based upon the exercise you can see that it is easy to program. We could use a program for which the function and its derivative must be specified in function statements. However, since the function will always be a polynomial, all that is really needed are the degree of polynomial and coefficients for each x term.

We present a subroutine for the Newton-Raphson method in Figure 6.4. This subroutine performs the necessary operations given the function, an initial starting point, and a stopping criterion. You must provide these latter items in a main program.

The construction of a main program requires considerable decision making. If you want to find all the real roots, it may be necessary to have many different starting points. This will also depend on the degree of the polynomial. In order to find all roots, you may have to run the program several times, using different starting points each time.

The main program listed in Figure 6.5 is flexible in that it allows the user to specify the number of starting points and their values. You may want to revise it to meet your

```
20600  REM ************************************************************
20601  REM  NEWTON-RAPHSON SUBROUTINE
20602  REM  SUBROUTINE TO PERFORM THE NEWTON-RAPHSON METHOD
20603  REM  TO FIND THE REAL ROOTS OF A POLYNOMIAL. REQUIRED INPUTS
20604  REM  ARE THE DEGREE OF THE POLYNOMIAL (JD), COEFFICIENTS FOR
20605  REM  ALL TERMS (STORED IN THE VECTOR F(I)), AN INITIAL
20606  REM  STARTING POINT (X), AND A STOPPING CRITERION (Z)
20607  REM
20608 JJ = 1
20609 X1 = X: REM  SAVE THE STARTING VALUE
20610 SUM = 0: REM  INITIALIZE THE FUNCTION
20611 PSUM = 0: REM  INITIALIZE THE DERIVATIVE
20612  FOR I = 0 TO JD
20613 SUM = SUM + (F(I) * (X ^ I)): REM  SUM TO FIND FUNCTION VALUE
20614  IF I = 0 THEN  GOTO 20616
20615 PSUM = PSUM + (F(I) * I * (X ^ (I - 1))): REM    SUM TO FIND DERIVATIVE VALUE
20616  NEXT I
20617  IF  ABS (SUM) <  = Z THEN  GOTO 20625: REM  FUNCTION MEETS STOPPING CRITERION
20618  IF PSUM = 0 THEN  GOTO 20623: REM  ZERO SLOPE
20619 X = X - (SUM / PSUM): REM  FIND THE NEXT POINT TO EVALUATE
20620 JJ = JJ + 1
20621  IF JJ > 100 THEN  GOTO 20627: REM  ITERATION LIMIT HAS BEEN REACHED
20622  GOTO 20610
20623  PRINT "FOR STARTING POINT ";X1;" THE POINT ";X;" HAS A ZERO SLOPE"
20624  GOTO 20630
20625  PRINT "FOR STARTING POINT ";X1;" THE ROOT IS ";X
20626  GOTO 20630
20627  PRINT : PRINT
20628  PRINT "THE ROOT HAS NOT BEEN FOUND AFTER"
20629  PRINT "100 ITERATIONS": PRINT : PRINT : PRINT
20630  PRINT : PRINT : PRINT : RETURN
```

Figure 6.4—Newton-Raphson Subroutine

```
100  REM   NEWTON-RAPHSON MAIN PROGRAM:
110  REM   MAIN PROGRAM TO FIND THE ROOTS OF A POLYNOMIAL
120  REM   USING THE NEWTON-RAPHSON SUBROUTINE.  THE USER MUST INPUT
130  REM   THE FOLLOWING DATA:
140  REM       JD=THE DEGREE OF THE POLYNOMIAL
150  REM       F(I)=COEFFICIENTS OF EACH TERM OF THE POLYNOMIAL
160  REM       Z=STOPPING CRITERION
170  REM   ALSO REQUIRED ARE INITIAL STARTING POINTS, WHICH ARE PASSED
180  REM   ON TO THE NEWTON-RAPHSON SUBROUTINE.
190  REM
210  INPUT "ENTER THE DEGREE OF THE POLYNOMIAL ";JD
220  DIM F(JD): REM   DIMENSION THE COEFFICIENT VECTOR
230  INPUT "ENTER THE STOPPING CRITERION";Z
240  FOR I = 0 TO JD
250  PRINT "ENTER THE ";I;" ORDER COEFFICIENT"
260  INPUT F(I): REM   ENTER THE FUNCTIOIN
270  NEXT I
280  INPUT "SPECIFY A STARTING POINT";X
290  GOSUB 20600: REM   CALL THE NEWTON-RAPHSON SUBROUTINE
300  PRINT "WOULD YOU LIKE TO TRY ANOTHER STARTING POINT ?"
310  INPUT "ENTER Y--YES, N--NO";A$
320  IF A$ = "Y" THEN  GOTO 280
322  PRINT "WOULD YOU LIKE TO SOLVE FOR COMPLEX"
324  INPUT "ROOTS? ENTER Y--YES, N--NO";A$
326  IF A$ = "N" THEN  GOTO 330
328  GOSUB 20631: REM   CALL NEWTON-RAPHSON COMPLEX
330  END
```

Figure 6.5—Newton-Raphson Main Program

particular needs. The issue of complex roots included in the program is discussed in the next section.

EXAMPLE 2. Use the main program in Figure 6.5 to solve for the real roots of the following function.

$$f(x) = x^4 - 11x^3 - 3x^2 + x - 7$$

SOLUTION. Hopefully, your choice of starting points led to the following roots:

Starting Point	Root
0	−0.9653
10	11.2634

The remaining roots of the polynomial are complex roots.

Complex Roots

In many applications, for instance in circuit analysis, the complex roots of a polynomial are of importance. The program presented so far found only the real roots. In this section, a series of subroutines is included to allow for the solution of complex roots.

Every polynomial has roots, whether they be real or complex. An example of a function which has no real roots is the following:

$$f(x) = 2x^2 - x + 1$$

A graph of this function is presented in Figure 6.6. Notice that there is no point where the function intersects the x-axis; thus, no real root. The roots of this function consist of both a real part and an imaginary part, making them complex numbers. The roots are:

$$x_1 = 0.25 + 0.66144i$$

$$x_2 = 0.25 - 0.66144i$$

where $i = \sqrt{-1}$. x_2 is called the "conjugate" of x_1; the sign on the imaginary part is changed. This situation always holds. That is, if a particular complex number is a root of a function, its conjugate is also a root. Based upon this concept, it is easy to see that complex roots always occur in pairs. Therefore, for a polynomial of odd order, we know that there must exist at least one real root.

There exist many interesting properties of complex numbers which will not be discussed here. However, it is important that they be considered in any evaluation of polynomials due to their importance in many scientific and engineering disciplines.

The Newton-Raphson procedure can be used as discussed earlier for the complex domain. The only difference is that the iterative points may be complex. We say "may" be complex because they need not be. Obviously, a complex number with an imaginary part of zero is a real number. Therefore, in a search for complex roots, real roots may result.

Unfortunately, your Apple does not provide automatic complex number handling. At this time, no personal computer does. Thus, the complex manipulations must be performed using "brute-force." Ruchdeschel (1981) presents an excellent discussion of handling complex numbers in this manner. There are many intricacies involved which we do not discuss here. The major concept is to convert a complex number to a polar coordinate

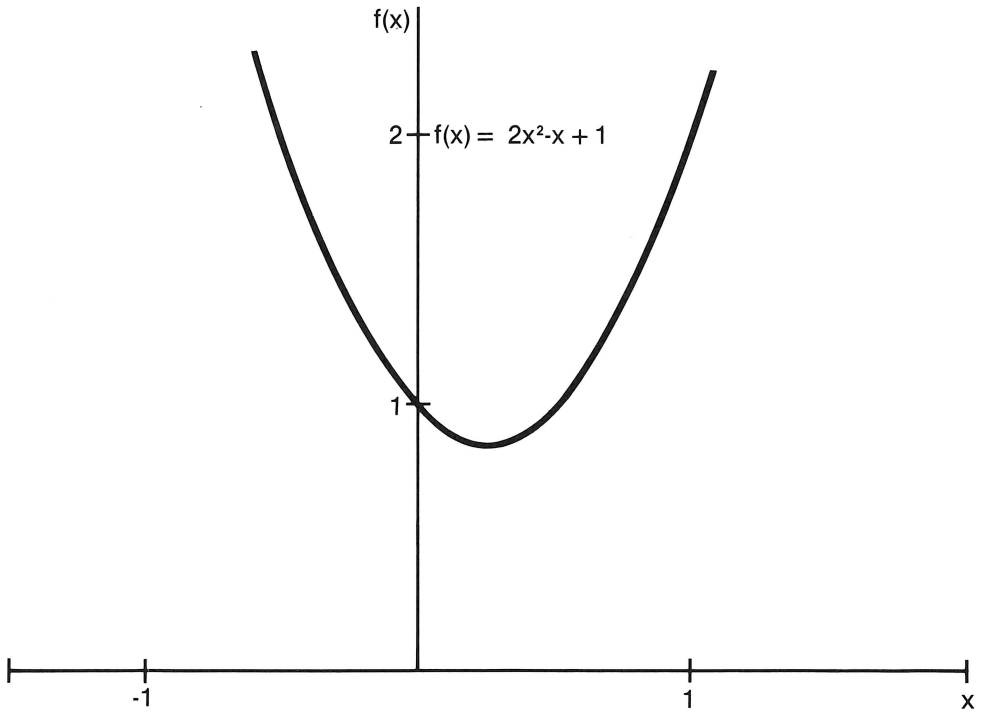


Figure 6.6—A Function With No Real Roots

representation, perform the necessary operations, and convert back to rectangular coordinates.

Figure 6.7 presents a series of subroutines to find (possibly) complex roots of a polynomial. You will notice that complex numbers have considerably complicated the procedure over the real numbers case. Special subroutines are included for summing the terms of the polynomial, summing the terms of the derivative of the polynomial, and dividing one complex number by another.

The main program in Figure 6.5 can be used to access this series of subroutines. The required starting point must contain both a real and an imaginary part. The root resulting from each complex starting point may be either real or complex. The root found is presented, along with its conjugate. Note, however, that because we are working with a specified stopping criterion, a complex root with a very small imaginary part may actually be a real root. This will depend upon the particular application.

EXAMPLE 3. Let us see how we can find the complex roots of the polynomial presented in Example 2. Use the main program in Figure 6.5.

$$f(x) = x^4 - 11x^3 - 3x^2 + x - 7$$

SOLUTION. We will examine the computer input and output as it appears on the screen. Both the real and complex analyses are performed.

```
20631  REM *************************
20632  REM  NEWTON-RAPHSON SUBROUTINE ---- COMPLEX ROOTS
20633  REM  SUBROUTINE TO PERFORM THE NEWTON-RAPHSON METHOD
20634  REM  TO FIND COMPLEX ROOTS OF A REAL POLYNOMIAL.  REQUIRED
20635  REM  INPUTS ARE THE ORDER OF THE POLYNOMIAL, JD, A STOPPING
20636  REM  CRITERION, Z, AND THE I(TH) ORDER COEFFICIENTS, IN THE
20637  REM  ARRAY F(I).  THREE ADDITIONAL SUBROUTINES ARE ACCESSED
20638  REM  FROM THIS SUBROUTINE TO PERFORM NECESSARY COMPLEX
20639  REM  NUMBER OPERATIONS.
20640  PRINT "ENTER THE REAL PART OF YOUR"
20641  INPUT "STARTING POINT  ";XR: PRINT
20642  PRINT "ENTER THE IMAGINARY PART OF YOUR"
20643  INPUT "STARTING POINT  ";XI: PRINT
20644 X1 = XR: REM  SAVE REAL PORTION OF STARTING POINT
20645 X2 = XI: REM   SAVE IMAGINARY PORTION OF STARTING POINT
20646  REM  INITIALIZE VARIABLES
20647 SR = 0
20648 SI = 0
20649 PR = 0
20650 PI = 0
20651  FOR I = 0 TO JD
20652  IF F(I) = 0 THEN  GOTO 20656
20653  GOSUB 20690: REM  CALCULATE THIS TERM FOR SR AND SI
20654  IF I = 0 THEN  GOTO 20656: REM  DERIVATIVE=0 FOR THIS TERM
20655  GOSUB 20715: REM  CALCULATE THIS TERM FOR PR AND PI
20656  NEXT I
20657  REM  STOP IF POLYNOMIAL VALUE * ITS CONJUGATE <=Z
20658 SUM = SR * SR + SI * SI
20659  IF  ABS (SUM) <  = Z THEN  GOTO 20676
20660  REM  CHECK FOR ZERO SLOPE
20661 PSUM = PR * PR + PI * PI
20662  IF  ABS (PSUM) < .000001 THEN  GOTO 20672
20663  REM  FIND NEXT EVALUATION POINT
20664  GOSUB 20740: REM  FIND POLY/DPOLY
20665  IF VP = 0 THEN  GOTO 20689
20666  REM  SUBTRACT TO FIND THE NEXT POINT
20667 XR = XR - DR
20668 XI = XI - DI
20669 JJ = JJ + 1: REM  UPDATE  ITERATION COUNTER
20670  IF JJ > 100 THEN  GOTO 20683: REM  ITERATION LIMIT HAS BEEN REACHED
20671  GOTO 20647
20672  PRINT : PRINT
20673  PRINT "FOR STARTING POINT ";XI;" +";X2;" I"
20674  PRINT "THE POINT ";XR;" + ";XI;" I HAS ZERO SLOPE"
20675  GOTO 20686
20676  PRINT : PRINT
20677  PRINT "FOR STARTING POINT ";X1;" + ";X2;" I"
20678  PRINT "THE ROOT IS ";XR;" + ";XI;" I"
20679  PRINT
20680  PRINT "IT'S CONJUGATE IS ALSO A ROOT"
20681  PRINT "            ";XR;" - ";XI;" I"
20682  GOTO 20686
20683  PRINT : PRINT
20684  PRINT "THE ROOT HAS NOT BEEN FOUND"
20685  PRINT "AFTER 100 ITERATIONS": GOTO 20689
20686  PRINT : PRINT : PRINT "WOULD YOU LIKE TO TRY ANOTHER COMPLEX"
20687  INPUT "STARTING POINT? Y--YES, N--NO";A$
20688  IF A$ = "Y" THEN  GOTO 20640
20689  RETURN
20690  REM *************************
20691  REM  SUM SUBROUTINE --- COMPLEX NUMBERS
20692  REM  FINDS ONE TERM OF A POLYNOMIAL FOR POSSIBLY
20693  REM  COMPLEX ROOTS.  CALLED FROM NEWTON-RAPHSON
20694  REM  COMPLEX SUBROUTINE.  INPUTS ARE THE CURRENT SOLUTION
20695  REM  (REAL AND COMPLEX PARTS), XR AND XI, AND SUM TERMS
```

```
20696  REM  SR AND SI.
20697  REM
20698  REM  STEP 1: CONVERT TO POLAR COORDINATES
20699 V =  SQR (XR * XR + XI * XI): REM  SET VECTOR
20700  IF XR = 0 THEN XR = .1 ^ 25: REM  DO NOT DIVIDE BY ZERO
20701 A =  ATN (XI / XR): REM  SET ANGLE
20702  IF XR < 0 THEN A = A + 3.1415927
20703  IF A < 0 THEN A = A + 6.2831853
20704  REM  STEP 2: FIND X^I IN POLAR COORDINATES
20705 V = V ^ I
20706 A = I * A
20707 A = A - 6.2831853 *  INT (A / 6.2831853): REM  ASSURE ANGLE IN QUAD 1
20708  REM  STEP 3: CONVERT TO RECTANGULAR COORDINATES
20709 YR = V *  COS (A)
20710 YI = V *  SIN (A)
20711  REM  FIND THE POLYNOMIAL SUM TO THIS POINT
20712 SR = SR + F(I) * YR
20713 SI = SI + F(I) * YI
20714  RETURN
20715  REM ************************
20716  REM  PSUM SUBROUTINE --- COMPLEX NUMBERS
20717  REM  FINDS ONE TERM OF THE DERIVATIVE OF A POLYNOMIAL
20718  REM  FOR POSSIBLY COMPLEX ROOTS.  INPUTS ARE THE CURRENT
20719  REM  SOLUTION(XR AND XI), AND THE TERMS OF THE SUM(PR AND PI).
20720  REM  THIS SUBROUTINE IS CALLED FROM THE NEWTON-RAPHSON
20721  REM  COMPLEX SUBROUTINE.
20722  REM
20723  REM   STEP 1: CONVERT TO POLAR COORDINATES
20724 V =  SQR (XR * XR + XI * XI)
20725  IF XR = 0 THEN XR = .1 ^ 25: REM  DO NOT DIVIDE BY ZERO
20726 A =  ATN (XI / XR)
20727  IF XR < 0 THEN A = A + 3.1415927
20728  IF A < 0 THEN A = A + 6.2831853
20729  REM  STEP 2: FIND X^I IN POLAR COORDINATES
20730 V = V ^ (I - 1)
20731 A = (I - 1) * A
20732 A = A - 6.2831853 *  INT (A / 6.2831853)
20733  REM  STEP 3: CONVERT TO RECTANGULAR COORDINATES
20734 YR = V *  COS (A)
20735 YI = V *  SIN (A)
20736  REM  FIND THE DERIVATIVE TO THIS POINT
20737 PR = PR + I * F(I) * YR
20738 PI = PI + F(I) * I * YI
20739  RETURN
20740  REM ************************
20741  REM  COMPLEX DIVISION SUBROUTINE
20742  REM  THIS SUBROUTINE IS CALLED FROM THE NEWTON-RAPHSON
20743  REM  COMPLEX SUBROUTINE. INPUTS REQUIRED ARE THE NUMERATOR
20744  REM  AND DENOMINATOR IN REAL AND IMAGINARY PARTS.  THESE ARE
20745  REM  SR AND SI, PR AND PI, RESPECTIVELY.  THE SUBROUTINE
20746  REM  RETURNS DR AND DI AS REAL AND IMAGINARY RESULTS OF
20747  REM  THE DIVISION.
20748  REM
20749  REM  CONVERT TO POLAR COORDINATES
20750 VS =  SQR (SR * SR + SI * SI)
20751  IF SR = 0 THEN SR = .1 ^ 25: REM  DO NOT DIVIDE BY ZERO
20752 AS =  ATN (SI / SR)
20753  IF SR < 0 THEN AS = AS + 3.1415927
20754  IF AS < 0 THEN AS = AS + 6.2831853
20755 VP =  SQR (PR * PR + PI * PI)
20756  IF PR = 0 THEN PR = .1 ^ 25: REM   DO NOT DIVIDE BY ZERO
20757 AP =  ATN (PI / PR)
20758  IF PR < 0 THEN AP = AP + 3.1415927
20759  IF AP < 0 THEN AP = AP + 6.2831853
20760  REM  PERFORM DIVISION
20761  IF VP = 0 THEN  GOTO 20769: REM  AVOID DIVISION BY ZERO
```

```
20762 DV = VS / VP
20763 DA = AS - AP
20764  IF DA < 0 THEN DA = DA + 6.2831853
20765  REM  CONVERT TO RECTANGULAR COORDINATES
20766 DR = DV *  COS (DA)
20767 DI = DV *  SIN (DA)
20768  GOTO 20770
20769  PRINT "ERROR: DIVISION BY ZERO IN COMPLEX DIVISION SUB"
20770  RETURN
```

Figure 6.7—Newton-Raphson for Complex Numbers

```
ENTER THE DEGREE OF THE POLYNOMIAL?4
ENTER THE STOPPING CRITERION?.001
ENTER THE 0 ORDER COEFFICIENT
?-7
ENTER THE 1 ORDER COEFFICIENT
?1
ENTER THE 2 ORDER COEFFICIENT
?-3
ENTER THE 3 ORDER COEFFICIENT
?-11
ENTER THE 4 ORDER COEFFICIENT
?1
SPECIFY A REAL STARTING POINT?0

FOR STARTING POINT 0 THE ROOT IS -.9652548

WOULD YOU LIKE TO TRY ANOTHER REAL
STARTING POINT? ENTER Y--YES, N--NO?Y
SPECIFY A REAL STARTING POINT?10

FOR STARTING POINT 10 THE ROOT IS 11.26337

WOULD YOU LIKE TO TRY ANOTHER REAL
STARTING POINT? ENTER Y--YES, N--NO?N
WOULD YOU LIKE TO SOLVE FOR COMPLEX
ROOTS? ENTER Y--YES, N--NO?Y
ENTER THE REAL PART OF YOUR STARTING POINT1

ENTER THE IMAGINARY PART OF YOUR STARTING POINT?1

FOR STARTING POINT 1 + 1 I
THE ROOT IS .3509703 + .7216114 I

IT'S CONJUGATE IS ALSO A ROOT
            .3509703 - .7216114 I

WOULD YOU LIKE TO TRY ANOTHER COMPLEX
STARTING POINT? Y--YES, N--NO?N
```

We need go no further in our solution search. Four roots have been found, 2 real and 2 complex. No more roots exist.

Of course, the location of roots is dependent on the starting points chosen. The main program used here required the user to input each starting point. For particular applications you may want to create a main program which generates starting points for you. There

are definite advantages and disadvantages to this, but it is certainly a viable alternative to the procedure used here.

6.3 Summary

This chapter has presented a procedure for finding the roots of a polynomial. We saw that for polynomials of third order or less, closed-form analytic solutions are possible. For most higher-order polynomials, a numerical procedure is appropriate. The procedure presented here, Newton-Raphson, was shown to be a logical, readily applicable method. By varying starting points we can locate the approximate roots of most polynomials with relatively little effort.

References

Hildebrand, F. B. *Introduction to Numerical Analysis*, McGraw-Hill Inc., New York, N.Y., 1974.
Nielson, K. L. *Methods in Numerical Analysis*, The MacMillan Company, New York, N.Y., 1956.
Ruchdeschel, F. R. *BASIC Scientific Subroutines Vol. 1*, McGraw-Hill, Inc., New York, N.Y., 1981.

Exercises

1. In optimization problems, we often must find the maximum or minimum value of a certain function. One way to locate stationary points of a function is to find the value of the variable (or set of variables) which equate the derivative of the function to zero. In the single variable case, we can use the Newton-Raphson procedure to perform the latter task. Using the method discussed above, find the real stationary points of the following function. Which is the "maximum value?"

   $f(x) = 7x^5 - 2x^3 - x^2 + 3x + 6$

2. Find the real roots of the following polynomials.

   a. $f(x) = 4x^6 - 2x^3 + 3x^2 - 19x + 12$
   b. $f(x) = 9x^4 + 3x^3 - 11x^2$
   c. $f(x) = x^5 + x^4 - 15x^3 + 6x - 3$

3. Investigate the use of the Newton-Raphson procedure to find roots of functions which are not polynomials. Is this a viable procedure? What problems are inherent in this approach?
4. Consider the following single-variable, nonlinear simultaneous equations.

   $x^2 + x = 6$
   $x^2 + 2x = 3$

   Discuss how these might be solved simultaneously using the Newton-Raphson procedure. What possible stumbling blocks are there?

5. Find all roots of the following polynomials. Before solving, specify the minimum number of real roots you will find.

   a. $f(x) = 8x^9 - 6x^4 + 3x - 12$
   b. $f(x) = x^6 + 3x^5 - 16x^2 + 9$
   c. $f(y) = 23y^5 - y^4 - y^3 + 6y$
   d. $f(z) = 11z^6 + z^3 - 1$

7

Numerical Integration

Integrating functions is a procedure used frequently in many engineering and science disciplines. When no closed-form expression can be found for an integral, we resort to numerical methods. Even if a closed-form expression is obtainable, its complexity may drive us to the ease of a computer solution.

We present and discuss two numerical integration techniques in this chapter. Each method can present a fairly accurate approximation for most functions. Those presented are the trapezoidal rule and Legendre-Gauss quadrature.

First, let us refresh our memory about integration. This will enable us to understand the reasons behind the numerical procedures better.

7.1 Basics of Integration

If you think about your experiences with integration you will probably remember that you found the integral by checking to see if its derivative was equal to the function to be integrated. In essence, you look to differentiation to integrate. You are not alone. In fact, integration is defined as the inverse of differentiation. Closed-form expressions for integrals are found by verifying that their derivatives are equal to the integrands.

It is important to distinguish between definite and indefinite integrals. Indefinite integrals do not have limits of integration, making the use of numerical methods inappropriate. Therefore, we will be concerned only with definite integrals, where the limits of integration are specified.

The integral of a function over some region is (for a single-variable function) the area under the curve generated by that function within the region. This area may be positive or negative, depending on whether the curve lies above or below the axis of the independent variable. Thus, for any function $y = f(x)$, the integral

$$\int_a^b y\,dx = \int_a^b f(x)\,dx$$

represents the area under the curve generated by y between the limits $x = a$ and $x = b$. We will use numerical procedures to calculate this area.

7.2 Numerical Methods

In order to find the area under the curve, different numerical integration procedures use a variety of approaches to divide the interval into a larger number of small pieces and add the areas of each piece. The major task is to find the area of each piece. This can be accomplished in many ways, each leading to a different numerical technique. For most functions, the numerically derived result will be approximate.

In this chapter we present two different integration methods. Each has its own method of generating the shapes of the small pieces.

Trapezoidal Rule

The trapezoidal rule is one of the simplest numerical integration techniques. The general procedure is to divide the interval of integration into a large number of small subintervals, each resembling a trapezoid. We find the integral by adding the areas of all the trapezoids. This is particularly easy because the area of a trapezoid is easily found by

$$\text{area} = \tfrac{1}{2}(a + b)h$$

where a and b are the lengths of the two sides and h is the distance between them. Since each area is, in general, not a true trapezoid, the result found is only an approximation. Figure 7.1 illustrates the formation of each trapezoid.

We chose points x_0, x_1, etc. so that there is a constant difference between consecutive points, say h. The lengths of the sides for each subinterval i are $y_i = f(x_i)$ and $y_{i+1} = f(x_{i+1})$. Using the formula for the area of a trapezoid, we can find the approximate area under the curve for each subinterval as:

$$A_i = \tfrac{1}{2}(y_i + y_{i+1})h$$

For n such subintervals, we have

$$\int_{x=x_0}^{x=x_n} ydx = A_1 + A_2 + \ldots + A_n + E$$

where E is a term representing the error resulting from the trapezoidal approximations for each subinterval. Substituting and combining terms, we find the formula easily applicable to computer manipulation.

$$\int_{x=x_0}^{x=x_n} ydx = h(\tfrac{1}{2}y_0 + y_1 + y_2 + \ldots + y_{n-1} + \tfrac{1}{2}y_n) + E$$

You can see that the numerical approximation will be more accurate if a large number of subintervals is used. As h gets smaller, each subinterval is more closely approximated by a trapezoid, yielding a better solution. The trade-off for this accuracy is increased computation time.

Figure 7.2 presents the BASIC subroutine for the trapezoidal rule. Notice that the number of subintervals used is an input to the subroutine. In this manner, you can control the accuracy and computation time you desire. To see how the number of subintervals affects accuracy, we developed two main programs to utilize the trapezoidal rule. Figure 7.3 presents the first. In this program the number of subintervals is an input. If the general

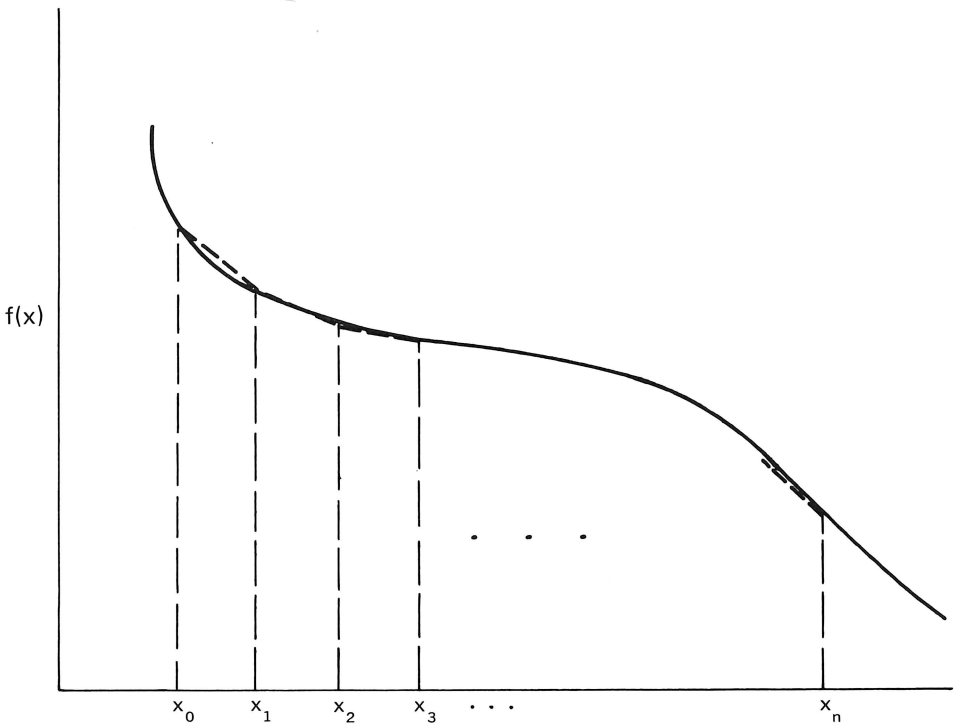


Figure 7.1—Numerical Integration Using the Trapezoidal Rule

shape of the function is known so you can properly judge the number of subintervals required, this can be the most efficient approach.

```
20900  REM ************************************************************
20910  REM  SUBROUTINE TO FIND THE INTEGRAL OF A FUNCTION USING
20920  REM  THE TRAPEZOIDAL RULE. INPUTS TO THIS SUBROUTINE MUST BE:
20930  REM           XBE:  LOWER INTEGRATION LIMIT
20940  REM           XEN:  UPPER INTEGRATION LIMIT
20950  REM           ISU:  NUMBER OF SUBINTERVALS USED
20960  REM     FUNCTION MUST BE SPECIFIED IN THE MAIN PROGRAM FN TRAP(X)
20970  TRAP = 0: REM  INITIALIZE INTEGRAL
20980  FOR I = 1 TO ISU - 1
20990  X = XBE + H * I: REM  FIND X VALUES; NOT INCLUDING ENDPOINTS
21000  TRAP + FN TRAP(X)
21010  NEXT I
21020  REM  CALCULATE END POINT VALUES, ADD TO TRAP
21030  TRAP = H * (TRAP + .5 * ( FN TRAP(XBE) + FN TRAP(XEN))): REM  RESULT
       IS THE INTEGRAL
21040  RETURN
```

Figure 7.2—Integration Trapezoidal Rule Subroutine

```
100  REM  TRAPEZOIDAL RULE: INTEGRATION:
110  REM  MAIN PROGRAM FOR NUMERICAL INTEGRATION USING
```

```
120  REM  THE TRAPEZOIDAL RULE. DATA REQUIRED ARE THE LIMITS
130  REM  OF INTEGRATION AND THE FUNCTION TO BE INTEGRATED,
140  REM  SPECIFIED IN LINE 150
150  DEF  FN TRAP(X) = EXP (X)
160  INPUT "THE LOWER LIMIT OF INTEGRATION IS ";XBE
170  INPUT "THE UPPER LIMIT OF INTEGRATION IS ";XEN
180  INPUT "THE NUMBER OF SUBINTERVALS TO BE USED IS ?";ISU
190  IF XBE > XEN THEN  GOTO 240
200  H = (XEN - XBE) / ISU: REM      CALCULATE EQUAL SPACING FOR SUBINTERVALS
210  GOSUB 20900: REM          CALL THE TRAPEZOIDAL RULE SUBROUTINE
220  PRINT "FOR ";ISU;" SUBINTERVALS THE INTEGRAL IS ";TRAP
230  GOTO 250
240  PRINT "XBE GREATER THAN XEN;PROGRAM TERMINATING"
250  END
```

Figure 7.3—Trapezoidal Rule Main Program

The example presented in Figure 7.3 is for the function $f(x) = e^x$. A sample result for specific inputs is presented in Figure 7.4. We can compare the results with the true closed-form solution. You can see that the number of subintervals significantly affects accuracy.

Number of Subintervals	Integral
5	1.724006
10	1.719713
20	1.71864
50	1.718339
100	1.718296

Actual: 1.718281828

Figure 7.4—Integration of EXP(X) Over 0–1

If computation time is not a concern, you may wish to specify a desired level of accuracy. We explain this level of accuracy as follows. The numerical approximation improves as h decreases, but beyond a certain point this improvement is negligible. If we find the integral a number of times, each time doubling the number of subintervals, we will reach a point where one solution is within a small amount, say z, of the previous solution. If z is quite small, on the order of 0.001, then you can feel confident in your solution. We present a main program for this type of analysis in Figure 7.5. Applying this to the same function examined earlier, f(x) = EXP(x), we get the results in Figure 7.6. Notice how the solution converges as the number of subintervals increases.

```
580  REM  TRAPEZOIDAL RULE: INTEGRATION: VARIABLE SUBINTERVALS
590  REM  MAIN PROGRAM FOR NUMERICAL INTEGRATION USING THE
600  REM  TRAPEZOIDAL RULE WITH INCREASING NUMBER OF SUBINTERVALS
610  REM   FUNCTION TO BE INTEGRATED MUST BE SPECIFIED IN LINE 640
620  REM  INPUTS FOR THIS PROCESS ARE LIMITS OF INTEGRATION, BEGINNING
630  REM  NUMBER OF SUBINTERVALS, AND STOPPING CRITERION Z.
640  DEF  FN TRAP(X) =  EXP (X)
650  INPUT "THE LOWER LIMIT OF INTEGRATION IS ? ";XBE
660  INPUT "THE UPPER LIMIT OF INTEGRATION IS ? ";XEN
670  INPUT "INITIAL # OF SUBINTERVALS TO BE USED IS ? ";ISU
680  INPUT "SPECIFY Z, THE STOPPING CRITERION ? ";Z
690  IL = 1: REM                    SET INITIAL PASS FLAG
```

```
700   H = (XEN - XBE) / ISU: REM       CALCULATE EQUAL SPACING FOR SPECIFIC ISU
710   IF XBE > XEN THEN  GOTO 800
720   GOSUB 20900: REM            CALL THE TRAPEZOIDAL RULE SUBROUTINE
730   PRINT "FOR ";ISU;" SUBINTERVALS THE INTEGRAL IS :": PRINT TRAP
740   IF IL = 1 THEN   GOTO 760
750   IF ABS (OLD - TRAP) <= Z THEN   GOTO 810
760   OLD = TRAP
770   IL = 0: REM                        CANCEL INITIAL PASS FLAG
780   ISU = ISU * 2: REM               DOUBLE THE NUMBER OF SUBINTERVALS
790   GOTO 700
800   PRINT "XBE IS > THAN XEN; PROGRAM TERMINATING"
810   END
```

Figure 7.5—Variable Subintervals Main Program

Initial Number of Subintervals:10
Stopping Criterion :0.0001

Subintervals	Integral
10	1.719714
20	1.71864
40	1.718371
80	1.718304

Figure 7.6—Results of One Run of Main Program in Figure 7.5

Legendre-Gauss Quadrature

Legendre-Gauss quadrature (integration) is somewhat more theoretically complex than the trapezoidal rule and considerably more accurate. The complex nature of all quadrature formulas (there are many) resulted in lack of use until the advent of advanced computing machinery. You can find many such techniques in Hildebrand (1974).

The major difference between any quadrature method and a "traditional" numerical integration approach such as the trapezoidal rule, is that the integration procedures involve ordinates (y-values) corresponding to equally spaced abscissas (x-values). Quadrature methods present an optimal distribution of the abscissas instead of arbitrarily specifying them. In this manner, the quadrature techniques can obtain approximately the same accuracy with fewer ordinates.

In general, these abscissas will be irrational. Weights by which the corresponding ordinates are multiplied may also be irrational. It is because of this that people were reluctant to use these procedures to solve problems by hand. We will not present the theory behind the development of Legendre-Gauss quadrature. You can find many fine texts which discuss this procedure in detail. You might want to examine Hildebrand (1974) Nelson (1956) or Scheid (1968). What is important here is using this technique to perform numerical integration.

There are actually many different forms for Legendre-Gauss quadrature, depending on the number of abscissas used in the approximation. The actual abscissas and their coefficients are derived from the Legendre polynomial. The three-point quadrature formula

will be used here. You can find abscissas and weights for other forms by formulas presented in Hildebrand (1974). We should note that accuracy increases as the number of abscissas increases. The three-point formula is:

$$\int_{-1}^{1} f(x)dx = \frac{1}{9}\left(5f\left(-\frac{\sqrt{15}}{5}\right) + 8f(0) + 5f\left(\frac{\sqrt{15}}{5}\right)\right) + E$$

You will notice that this formula is defined over the interval -1 to 1. A change of variables can be used to transform your integral to meet this requirement. We can include such a change in a general formula.

$$\int_{a}^{b} f(x)dx = \frac{b-a}{18}(5f(y - z) + 8f(y) + 5f(y + z)) + E$$

where,

$$y = (a + b)/2$$

$$z = [(b - a)/2]\sqrt{3/5}$$

The accuracy of this quadrature formula is very good. In fact, the three-point formula is exact for polynomials of order five or less. You can use the formula as it is or divide a large interval into a number of subintervals, each one evaluated via Legendre-Gauss. These areas would then be summed.

We present a subroutine for Legendre-Gauss quadrature in Figure 7.7. The main program for accessing this subroutine is given in Figure 7.8 with the function $f(x) = 2x^2 + 2x$. If you run this main program you can see that the result is exact, as was expected.

```
21050  REM ************************************************************
21060  REM  LEGENDRE-GAUSS QUADRATURE SUBROUTINE:  INTEGRATION:
21070  REM  SUBROUTINE FOR NUMERICAL INTEGRATION USING
21080  REM  LEGENDRE-GAUSS QUADRATURE. INPUTS ARE THE FUNCTION
21090  REM   TO BE INTEGRATED, FN LG(X), SPECIFIED IN A FUNCTION STATEMENT,
21100  REM  LOWER INTEGRAL LIMIT A, AND UPPER LIMIT B, RETURNS LGS
21110  Y = (A + B) / 2: REM  CALCULATE THE INTERVAL MIDPOINT
21120  Z = ((B - A) / 2) *  SQR (3 / 5)): REM  CALCULATE Z FOR REMAINING
       ABSCISSAS
21130  LGS = ((B - A) / 18) * (5 *  FN LG(Y - Z) + 8 *  FN LG(Y) + 5 *
       FN LG(Y + Z)): REM   L-G FORMULA
21140  RETURN
```

Figure 7.7—Legendre-Gauss Quadrature Subroutine

```
270  REM  MAIN PROGRAM FOR LEGENDRE-GAUSS QUADRATURE
280  REM  INPUTS ARE THE FUNCTION ITSELF, SPECIFIED IN LINE 300
290  REM  AND THE LIMITS OF THE INTEGRATION.
300  DEF  FN LG(X) = 2 * (X^ 2) + 2 * X: REM      INPUT THE FUNCTION TO
     BE INTEGRATED
320  INPUT "THE LOWER LIMIT OF INTEGRATION IS ? ";A
320  INPUT "THE UPPER LIMIT OF INTEGRATION IS ";B
330  IF A > B THEN  GOTO 370
340  GOSUB 21050: REM          CALL THE LEGENDRE-GAUSS SUBROUTINE
350  PRINT "THE AREA UNDER THE CURVE FROM ";A;" TO ";B;" IS ";LGS
360  GOTO 380
370  PRINT "LOWER LIMIT > UPPER LIMIT; PROGRAM TERMINATING"
380  END
```

Figure 7.8—Legendre-Gauss Main Program

EXAMPLE. Write a main program to integrate EXP(X) over the interval 0 to 1 by using Legendre-Gauss quadrature for each of 25 subintervals of length 0.04.

SOLUTION. The main program for this procedure is given in Figure 7.9. Notice how the interval is divided into 25 subintervals. If you compare the result with that found using the trapezoidal rule, you can see how accurate the Legendre-Gauss procedure is for fewer functional evaluations. From the computer run: area under the curve is 1.71828183.

```
400  REM  SOLUTION TO EXAMPLE PROBLEM 2; LEGENDRE-GAUSS OVER 0-1
410  REM  WITH SUBINTERVALS OF SIZE 0.04. FUNCTION EXP(X)
420  DEF  FN LG(X) =  EXP (X): REM          FUNCTION TO BE EVALUATED
430  AREA = 0: REM                        INITIALZE
440  W =  SQR (3 / 5): REM                  INITIALIZE
450  FOR I = 0 TO 24
460  A = .04 * I
470  B = A + .04
480  K = (B - A) / 2
490  Z = W * K: REM                  SET Z VALUE FOR ABSCISSAS
500  Y = (A + B) / 2: REM            CALCULATE MIDPOINT OF SUBINTERVAL
510  GOSUB 21050: REM          CALL LEGENDRE-GAUSS SUBROUTINE
520  AREA = LGS + AREA: REM       CALCULATE THE CUMULATIVE AREA
530  NEXT I
540  PRINT : PRINT "AREA UNDER THE CURVE IS ";AREA
550  END
```

Figure 7.9—Main Program for Example Problem

7.3 Summary

In this chapter we have discussed the operational aspects of numerical integration. We first briefly presented the concepts of integration. Sometimes closed-form solutions are not possible or finding them is not feasible. In these cases, numerical procedures can be implemented on your Apple Computer to approximate the integral.

Two techniques were presented: the trapezoidal rule and Legendre-Gauss quadrature. The trapezoidal rule is easy to understand and works reasonably well when small subintervals are used. The Legendre-Gauss quadrature is much more complex, but also more accurate. Subroutines for each of these procedures are included. You may wish to consult the references listed below for additional numerical integration techniques.

References

Hildebrand, F. B. *Introduction to Numerical Analysis*, McGraw-Hill Book Co., New York, N.Y., 1974.

Nelson, K. L. *Methods in Numerical Analysis*, The Macmillan Publishing Co., New York, N.Y., 1956.

Scheid, Francis, *Theory and Problems of Numerical Analysis* Schaum's Outline Series, McGraw-Hill Book Co., New York, N.Y. 1968.

Exercises

1. Integrate the following functions over the interval 1 to 3 using the trapezoidal rule with $h = 0.01$ and Legendre-Gauss quadrature applied over two subintervals. Compare the results. Which value of h makes the two methods roughly equivalent?

   a. $f(x) = 8x^3 - 9x + 3$

   b. $f(x) = e^{3x} + \sqrt{x}$

2. How do the integration methods presented here handle functions which dip below the x-axis? Are revisions required? Explain.

3. In statistics, the area between two reference points under the curve generated by a probability density function is equal to the probability of a point lying between the two reference points. One such density function is the "normal" density function. It is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \qquad -\infty < x < \infty$$

   where

   μ = mean of the distribution

   σ = standard deviation of the distribution

   From this information, find the following probabilities:

   a. Probability $(2 \leq X \leq 5)$ for $\mu = 4$, $\sigma = 0.5$

   b. Probability $(0.1 \leq X \leq 0.5)$ for $\mu = 0.7$, $\sigma = 0.3$

   c. Probability $(70 \leq X \leq 85)$ for $\mu = 80$, $\sigma = 12$

4. Suppose we want to find only the positive area under a curve. Revise the trapezoidal rule subroutine to handle this and illustrate it on the following functions.

   a. sin x from 1 to 6

   b. $x^2 - 4x + 1$ from 0 to 5

5. Find the integral of the following functions

   a. $f(x) = e^{x^2-2} + x^2$ over 0 to 1

   b. $f(x) = \sin x - \cos x + x^3$ over 0 to 2

   c. $f(x) = x^2 \cos x + x^3$ over 1 to 3

6. (Advanced topic) For functions of more than one variable, we are concerned with multiple integration. How would the procedures described in this chapter be adapted to address multiple integrals? Illustrate using an example problem.

8

Numerical Solutions to Differential Equations

When analyzing physical systems, we find that differential equations occur naturally quite frequently. The solution to many simple forms of differential equations can be easily obtained by hand. In many cases, however, a straightforward solution is not possible. As in other situations when a closed-form solution cannot be used, or a computer solution is desired, we resort to numerical methods. In this chapter we present such a procedure to obtain approximate solutions to first-order ordinary differential equations. The procedure can be extended to solve systems of differential equations.

8.1 Overview of Differential Equations

A differential equation is an equation containing derivatives of an unknown function. In general, the solution of the differential equation requires determination of the unknown function, g(x). Many important problems engineers and scientists face are formulated as differential equations. Some typical differential equations occurring in physical systems are the wave equation in fluid mechanics, the potential equation in electricity, and the diffusion equation in elasticity.

The aforementioned differential equations happen to be partial differential equations. These have terms of partial derivatives rather than ordinary derivatives. The latter case is represented by ordinary differential equations. Boyce and DiPrima (1977) present two excellent examples of ordinary differential equations representing physical systems. The first relates the charge on a condenser, Q(t), in a circuit with capacitance C, resistance R, inductance L, and impressed voltage E(t).

$$L\frac{d^2Q(t)}{dt^2} + R\frac{dQ(t)}{dt} + \frac{Q(t)}{C} = E(t) \qquad (1)$$

The second represents the decay of an amount R(t) of a radioactive substance over time, where k is a known constant.

$$\frac{dR(t)}{dt} = -kR(t) \tag{2}$$

The equations (1) and (2) above are different types of differential equations. Equation (1) is a second-order differential equation and equation (2) is a first-order differential equation. The order is that of the highest derivative that appears in the equation. We will be concerned only with the first-order case.

The general form of the ordinary differential equations we will be examining is given as follows:

$$y' = f(x,y) \qquad y(x_0) = y_0 \tag{3}$$

This is called an initial value problem, since the second equation above prescribes the point (x_0, y_0) through which the integral curve passes, thus specifying an initial condition.

Solving a problem represented by equations (3) analytically cannot be reduced to a general methodology. Solution procedures are available only for special classes of equations. A determination of the solution must begin with a determination of the class of equations to which the problem of interest belongs. For some of these classes, standard solution forms are available. As an example, consider the following differential equation:

$$y' + ay = 0 \qquad \text{a is a real constant.} \tag{4}$$

The solution to this can be expressed as

$$y = ce^{-ax} \qquad \text{c an arbitrary constant.} \tag{5}$$

With the specification of an initial condition, the value of c can be assessed.

EXAMPLE 1. Solve the following initial value problem analytically.

$$y' + 5y = 0 \qquad y(0) = 2$$

SOLUTION. You can see that this differential equation is of the same form as equation (4) and the solution is therefore specified in equation (5).

$$y = ce^{-ax} = ce^{-5x}$$

To find the value of c, we apply the initial value condition

$$y(0) = ce^{-5(0)} = c = 2$$

The final result is

$$y = 2e^{-5x}.$$

Solving these problems analytically is often an interesting exercise, but we are really not concerned with that here. We, and you, want to address the solution of differential equations on the computer. A discussion of solving these via numerical methods is presented in the next section.

8.2 Numerical Solutions

As we have discussed earlier, there may be several reasons to prefer a computerized numerical solution to an analytic solution. A straightforward closed-form solution may not be obtainable, an available analytic procedure may be long and complex, or we simply may not want to expend the necessary energy to determine the analytic solution. There happen to be some powerful numerical techniques available to approximate the solution to differential equations that we can use in these types of situations. These methods have become more popular as their ease of application increases with the power of the computer.

We need to make some assumptions prior to presenting the numerical procedure. First, we will be considering first-order ordinary differential equations of the form given in equation (3). We must also assume that there exists a unique solution to the problem in some interval surrounding x_0. In order to draw this conclusion, we must assume that f(x,y) and its partial derivative with respect to y are continuous in a rectangular region containing (x_0, y_0). These assumptions may not always be straightforward to verify, but this will not prevent us from attempting to find a solution. We must, however, be very careful in interpreting our result. If the required assumptions do not hold, our result could be very far from the correct solution.

The numerical method we present, known as the Runge-Kutta method, represents a class of methods called one-step or starting methods. Another group of methods, called multistep or continuing methods, is discussed in Boyce and DiPrima (1977).

The Runge-Kutta method was developed initially by Carl Runge in 1895 and continued by M. W. Kutta in 1901. The procedure is based on the Taylor-series expansion of the function y. This expansion is

$$y_{n+1} = y_n + hy_n' + \frac{h^2}{2} y_n'' + \frac{h^3}{6} y_n''' + \frac{h^4}{24} y_n^{iv} + \cdots \tag{6}$$

We will consider the Runge-Kutta formula found by truncating the expansion following the terms presented above. An approximation is used for calculating y using the following formula:

$$y_{n+1} = y_n + h[a_0 f(x_n,y_n) + a_1 f(x_n + b_1 h, y_n + c_1 h) + a_2 f(x_n + b_2 h, y_n + c_2 h) + a_3 f(x_n + b_3 h, y_n + c_3 h)]$$

The a's, b's, and c's are determined such that the coefficients agree with the corresponding coefficients of equation (6). Notice that this latter expression does not require the evaluation of any derivatives. This is an important advantage. There are several evaluations of f(x,y), but this is no problem for computer application.

By using this type of approximation, we can formulate the Runge-Kutta formula as follows:

$$y_{n+1} = y_n + \frac{h}{6}(k_0 + 2k_1 + 2k_2 + k_3) \tag{7}$$

where

$$k_0 = f(x_n, y_n)$$

$$k_1 = f(x_n + 0.5h, y_n + 0.5hk_0)$$

$$k_2 = f(x_n + 0.5h, y_n + 0.5hk_1)$$

$$k_3 = f(x_n + h, y_n + hk_2)$$

Because of the number of terms used from the Taylor-series expansion, this is called a fourth-order Runge-Kutta formula.

Throughout the discussion so far, we have assumed a uniform step size h on the x-axis. That is, $x_n = x_0 + nh$. This step size, or spacing, has an impact on the accuracy of our approximation. In fact, the error in using the formula of equation (7) is proportional to h, and for some interval the formula has an accumulated error of, at most, a constant times h (Boyce and DiPrima, 1977). Thus, we can control our error through selection of the step size h. Remember, though, that this increased accuracy comes at the expense of increased computation time. We will not discuss the error term here, but you may find it worthwhile to consult the references at the end of the chapter in this regard.

At this time, let us examine a sample problem to see how the procedure works in actually solving a differential equation.

EXAMPLE 2. Solve the following differential equation using the Runge-Kutta formula from equation (7) at $x = 0.2$ and $h = 0.2$.

$$y' - 2xy = x \qquad y(0) = 1$$

SOLUTION. We first rewrite the differential equation in standard form.

$$y' = x + 2xy \qquad y(0) = 1$$

We can now immediately begin the solution procedure by recognizing from the initial condition that $x_0 = 0$, $y_0 = 1$. First calculate the k values.

$$k_0 = f(x_0, y_0) = f(0, 1) = 0$$

$$k_1 = f(x_0 + 0.5h, y_0 + 0.5hk_0) = f(0 + 0.1, 1 + (0.1)(0)) = 0.3$$

$$k_2 = f(x_0 + 0.5h, y_0 + 0.5hk_1) = f(0 + 0.1, 1 + (0.1)(0.3)) = 0.306$$

$$k_3 = f(x_0 + h, y_0 + hk_2) = f(0 + 0.2, 1 + (0.2)(0.306)) = 0.62448$$

Plugging these into equation (7) yields:

$$y = 1 + \frac{0.2}{6}(0 + (2)(0.3) + (2)(0.306) + 0.62448) = 1.061216$$

This result tells us that the approximate value of the function $y = g(x)$ (the solution to the differential equation) evaluated at $x = 0.2$ is 1.061216. It is interesting to note that this corresponds to the actual value found from the true function g(x). This function is

$$y = g(x) = -\frac{1}{2} + \frac{3}{2}e^{x^2}.$$

If we evaluate this function at $x = 0.2$, we find

$$g(0.2) = -\frac{1}{2} + \frac{3}{2}e^{(0.2)^2} = 1.06121616$$

We can improve the accuracy of the approximation by decreasing the step size. In the previous example this hardly seems to be necessary due to the accuracy of the result. This will not always be the case, however. Though computation time may increase, the increase in accuracy may be worthwhile.

How do we effect an increase in the step size operationally? The Runge-Kutta method presents us with a value of x and an associated value of $y = g(x)$. Remember that h is a uniform spacing along the x-axis. We specify successive values of x by using the following relationships:

$$x_n = x_0 + nh \tag{8}$$

$$x_n = x_{n-1} + h \tag{9}$$

In the previous example we used a value of $h = 0.2$. Since $x_0 = 0$ in this case, we would have the following sequence of x values for successive iterations throughout the procedure.

n	x	value
0	x_0	0
1	$x_1 = 0+h$	0.2
2	$x_2 = 0.2+h$	0.4
3	$x_3 = 0.4+h$	0.6
•	•	•
•	•	•
•	•	•

Thus, if we made another pass through the formula in Example 2, starting where we left off, we would find the approximate value of $y = g(x)$ at $x = 0.4$. Another pass would lead to g(x) at $x = 0.6$, and so on. By examining equation (8) we can determine easily the appropriate step size h required for a particular number of iterations to the desired point at which to evaluate g(x).

For example, suppose $x_0 = 1$ and we are interested in the solution of a differential equation at the point $x_n = 5$. If we wanted to use 20 iterations through the Runge-Kutta formula, we would solve equation (8) for h to find the step size.

$$h = \frac{x_n - x_0}{n} = \frac{5-1}{20} = 0.2$$

When selecting the step size, it is important to remember that the desired point of evaluation, x_n, must be an integer multiple of h from the initial value x_0.

Now you know how the Runge-Kutta method works. In the next section, we will examine the computer application of this procedure.

8.3 Application of the Runge-Kutta Method

Through Example 2 you can see that application of the Runge-Kutta method is not complex and should be easily performed by your Apple Computer. The repeated functional evaluations make this a natural for computer implementation.

The general solution procedure is specified below. The computer program follows these same steps.

1. Specify f(x,y) from the problem statement.
2. Choose an appropriate step size h for the problem.
3. Begin the procedure at x_0 and y_0, which are specified as the initial condition of the problem.
4. Calculate k_0, k_1, k_2, and k_3 terms based upon current values for x_n and y_n.

```
21200  REM ************************************************************
21210  REM  RUNGE-KUTTA:  FIXED STEPSIZE
21220  REM  SUBROUTINE TO PERFORM THE RUNGE-KUTTA METHOD FOR
21230  REM  SOLVING DIFFERENTIAL EQUATIONS. INPUT TO THE
21240  REM  SUBROUTINE MUST BE :
21250  REM       1. STEPSIZE TO BE USED: H
21260  REM       2. INITIAL CONDITION VALUE FOR X: R
21270  REM       3. INITIAL CONDITION VALUE FOR Y(X): S
21280  REM       4. STOPPING POINT ALONG X-AXIS: MAX
21290  REM       5. FUNCTION SPECIFIED IN LINE 22650
21300 L =  INT ((MAX - R) / H): REM    NUMBER OF REQUIRED FVALUATION POINTS
21310  DIM X(L + 2),Y(L + 2): REM   SET DIMENSIONS
21320 I = 0
21330 X(0) = R: REM               SET INITIAL CONDITION
21340 Y(0) = S: REM               SET INITIAL CONDITION
21350  PRINT : PRINT : PRINT
21360  PRINT "RUNGE-KUTTA PROCEDURE FOR STEPSIZE= ";H
21370  PRINT
21380  PRINT "STEP             X         Y=G(X)"
21390  PRINT "=========================================="
21400  PRINT  TAB( 2);I;
21410  PRINT  TAB( 18);
21420 X(I) =  INT (X(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
21430  PRINT X(I);
21440  PRINT  TAB( 26);
21450 Y(I) =  INT (Y(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
21460  PRINT Y(I)
21470 X(I + 1) = X(I) + H
21480  IF X(I + 1) > MAX THEN  GOTO 21690
21490 IH = 0
21500 IJ = 0
21510 XX = X(I)
21520 YY = Y(I)
21530  GOSUB 22630
21540 IJ = 1
21550 XX = X(I) + .5 * H
21560 YY = Y(I) + .5 * H * K
21570  GOSUB 22630
21580 IJ = 2
21590 XX = X(I) + .5 * H
21600 YY = Y(I) + .5 * H * K1
21610  GOSUB 22630
21620 IJ = 3
21630 XX = X(I) + H
21640 YY = Y(I) + H * K2
21650  GOSUB 22630
21660 Y(I + 1) = Y(I) + (H / 6) * (K + 2 * K1 + 2 * K2 + K3)
21670 I = I + 1
21680  GOTO 21400
21690  RETURN
```

Figure 8.1—Runge-Kutta Subroutine

5. Find the value of $y_{n+1} = g(x_{n+1})$ from equation (7).

6. If more function evaluations are desired, return to step 4; otherwise stop.

This procedure is not difficult to program. A subroutine to perform the Runge-Kutta method is presented in Figure 8.1. Since the differential equation consists of two variables, a special subroutine is required. This subroutine is presented in Figure 8.4. The differential equation must be specified in this subroutine. We can easily write a main program to obtain input information for the Runge-Kutta subroutine. This is illustrated in the following example.

EXAMPLE 3. Write a main program to utilize the subroutine in Figure 8.1 and use it to solve the following differential equation at x = 0.2, 0.4, and 0.5.

$$y' + 2xy - 3x = 0 \qquad y(0) = 1$$

SOLUTION. A main program to collect input data for a differential equation is presented in Figure 8.2. Notice that the function must be specified within the subroutine in Figure 8.4. Line 22650 specifies the differential equation. Inputs include the step size h, the initial conditions, and the last x value to be evaluated.

```
100 REM  RUNGE-KUTTA MAIN PROGRAM
110 REM  MAIN PROGRAM TO UTILIZE THE RUNGE-KUTA
120 REM  SUBROUTINE. ACCESS TO THE FIXED STEP SIZE
130 REM  SUBROUTINE OR THE REVISED (VARIABLE STEP SIZE)
140 REM   SUBROUTINE IS POSSIBLE.
150 INPUT "ENTER THE DESIRED OR INITIAL STEP SIZE?";H
160 PRINT : PRINT
170 PRINT "ENTER THE INITIAL CONDITIONS FOR
Y(X0)=(Y0)"
180 PRINT
190 INPUT "ENTER THE INITIAL VALUE X0 ? ";R
200 INPUT "ENTER THE INITIAL VALUE Y0 ? ";S
210 PRINT : PRINT
220 PRINT "SPECIFY THE LAST X - VALUE AT WHICH TO  EVALUATE
G(X)"
230 PRINT "NOTE: IF THIS VALUE IS NOT AN INTEGER"
240 PRINT "MULTIPLE OF H FROM X0, IT WILL NOT BE"
250 PRINT "EVALUATED
260 INPUT MAX
270 REM  INPUT THE DIFFERENTIAL EQUATION TO BE SOLVED
280 GOSUB 21200: REM                STEP SIZE SUBROUTINE
290 END
```

Figure 8.2—Runge-Kutta Main Program

From the problem statement we must obtain evaluations for x = 0.2, 0.4, and 0.5. Since there are different intervals between these points we could proceed in many different ways. Using a step size of h = 0.1, we would find g(x) for x = 0.1, 0.2, 0.3, 0.4, 0.5, which would meet the requirements. We could also use a step size of h = 0.2 for x = 0.2 and 0.4, and a step size of 0.5 for x = 0.5. Using the main program in these two cases, we get the results presented below.

Step Size	X	Y = G(X)
0.1	0	1
	0.1	1.00498
	0.2	1.01961
	0.3	1.04304
	0.4	1.07393
	0.5	1.11060
0.5	0	1
	0.5	1.11068

Using data found from the Runge-Kutta method for various values of x, one could approximate the function $y = g(x)$ within that range. Chapter 4 discusses some curve fitting techniques which might be used.

The Runge-Kutta procedure can be made more reliable and efficient through automatic adjustment of the step size during computation. You may desire a certain accuracy in the approximation during each step. By specifying this desired accuracy, you allow the program to adjust the step size to ensure that the accuracy is being maintained. If the error is too large at any point, the step size is automatically decreased in order to increase accuracy. If the error is extremely small, the step size can be increased in order to improve efficiency. This type of procedure is included in the subroutine presented in Figure 8.3. This procedure accepts an initial step size from you, the user. An acceptable error must also be specified. The program calculates each point using both the specified step size, and one-half of that value, and compares them. If the difference between these values is greater than the acceptable error, the step size is cut in half. If the difference is less than the acceptable error divided by 100, the step size is doubled. In any other case, the step size remains the same.

EXAMPLE 4. Solve the differential equation in Example 3 for $x = 0.1, \ldots, 0.8$ using an initial step size of 0.1 and a tolerance of 0.01, using the subroutine in Figure 8.3.

$$y' + 2xy - 3x = 0 \qquad y(0) = 1$$

SOLUTION. Using the main program in Figure 8.2 to access the variable-step size subroutine, we obtain the results illustrated below. Notice how this procedure may not yield the specific results we need.

Step	Step Size	X	Y = G(X)
0	0.100	0.00000	1.00000
1	0.100	0.10000	1.00498
2	0.200	0.30000	1.04303
3	0.400	0.70000	1.19373

8.4 Summary

The numerical procedure presented in this chapter, Runge-Kutta, can prove quite useful in finding the solution to differential equations. Remember that we have considered only first-order ordinary differential equations. The procedure can be revised to solve a series of differential equations (see Hildebrand, 1974). The Runge-Kutta method allows us to solve many problems occurring naturally in the physical environment.

```
21700  REM ************************************************************
21710  REM  RUNGE-KUTTA:  VARYING STEPSIZE
21720  REM  SUBROUTINE TO PERFORM THE RUNGE-KUTTA METHOD FOR
21730  REM  SOLVING DIFFERENTIAL EQUATIONS. INPUT TO THE
21740  REM  SUBROUTINE MUST BE:
21750  REM            1. INITIAL STEPSIZE TO BE USED: H
21760  REM            2. INITIAL CONDITION VALUE FOR X: R
21770  REM            3. INITIAL CONDITION VALUE FOR Y(X): S
21780  REM            4. STOPPING POINT ALONG X-AXIS: MAX
21790  REM             5. FUNCTION SPECIFIED IN LINE 22650
21800  REM   THIS SUBROUTINE WILL ASK FOR THE SPECIFICATION OF A
21810  REM   DESIRED TOLERANCE AT EACH STEP, T
21820  REM
21830  INPUT "ENTER DESIRED TOLERANCE AT EACH STEP";T
21840 L =  INT ((MAX - R) / H): REM     NUMBER OF REQUIRED EVALUATION POINTS
21850  DIM X(L + 1),Y(L + 1): REM       SET DIMENSIONS
21860 I = 0
21870 J = 1: REM         SET "SAME OR HALF" STEPSIZE FLAG
21880 X(0) = R: REM      SET INITIAL CONDITION
21890 Y(0) = S: REM      SET INITIAL CONDITION
21900  PRINT : PRINT : PRINT
21910  PRINT "RUNGE-KUTTA PROCEDURE FOR STEPSIZE= ";H
21920  PRINT
21930  PRINT "STEP   STEPSIZE      X           Y=G(X)"
21940  PRINT "========================================="
21950  PRINT  TAB( 2);I;
21960  PRINT  TAB( 10);
21970 H =  INT (H * 10 ^ 3 + .5) /  INT (10 ^ 3 + .5)
21980  PRINT H;
21990  PRINT  TAB( 22);
22000 X(I) =  INT (X(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
22010  PRINT X(I);
22020  PRINT  TAB( 33);
22030 Y(I) =  INT (Y(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
22040  PRINT Y(I)
22050  IF J = 2 THEN H = H * 2: REM   DOUBLE STEPSIZE
22060 J = 1: REM     RESET FLAG
22070 X(I + 1) = X(I) + H
22080  IF X(I + 1) > MAX THEN  GOTO 22610
22090 IH = 0
22100 IJ = 0
22110 XX = X(I)
22120 YY = Y(I)
22130  GOSUB 22630
22140 IJ = 1
22150 XX = X(I) + .5 * H
22160 YY = Y(I) + .5 * H * K
22170  GOSUB 22630
22180 IJ = 2
22190 XX = X(I) + .5 * H
22200 YY = Y(I) + .5 * H * K1
22210  GOSUB 22630
22220 IJ = 3
22230 XX = X(I) + H
22240 YY = Y(I) + H * K2
22250  GOSUB 22630
22260 Y(I + 1) = Y(I) + (H / 6) * (K + 2 * K1 + 2 * K2 + K3)
22270  REM  HALVE THE STEP SIZE TO CHECK THE ACCURACY OF THE APPROXIMATION
22280 HH = H / 2
22290 YY = Y(I)
22300 XX = X(I)
22310  FOR II = 1 TO 2: REM    FIND THE NEXT Y FOR HALF THE STEPSIZE
22320 IH = 1
22330 IJ = 0
22340  GOSUB 22630
22350 IJ = 1
```

```
22360 XX = XX + .5 * HH
22370 YY = YY + .5 * HH * M
22380  GOSUB 22630
22390 IJ = 2
22400 XX = XX + .5 * HH
22410 YY = YY + .5 * HH * M1
22420  GOSUB 22630
22430 IJ = 3
22440 XX = XX + HH
22450 YY = YY + HH * M2
22460  GOSUB 22630
22470 YY = YY + (HH / 6) * (M + 2 * M1 + 2 * M2 + M3)
22480 XX = XX + HH
22490  NEXT II
22500  REM  CHECK TO SEE IF THE APPROXIMATION MEETS THE SPECIFIED TOLFRANCE
22510  IF  ABS (YY - Y(I + 1)) > T THEN  GOTO 22560: REM    DECREASE STEPSIZE
22520  IF  ABS (YY - Y(I + 1)) > T / 100 THEN  GOTO 22540: REM    INCREASE STEPSIZE
22530  GOTO 22590: REM     KEEP CURRENT STEPSIZE
22540 J = 2: REM        SET "DOUBLE STEPSIZE" FLAG
22550  GOTO 22590
22560  PRINT "TOLERANCE NOT MET ON THE FOLLOWING STEP: H HALVED"
22570 Y(I) = YY
22580 H = HH
22590 I = I + 1
22600  GOTO 21950
22610  RETURN
```

Figure 8.3—Runge-Kutta, Varying Step Size Subroutine

```
22620  REM  ************************************************
22630  REM  SUBROUTINE FOR FUNCTION EVALUATION
22640  REM       3XX-2XX * YY
22650 FCN = (3 * XX) - (2 * XX * YY)
22660  IF IH = 1 THEN  GOTO 22720
22670  IF IJ = 0 THEN K = FCN
22680  IF IJ = 1 THEN K1 = FCN
22690  IF IJ = 2 THEN K2 = FCN
22700  IF IJ = 3 THEN K3 = FCN
22710  RETURN
22720  IF IJ = 0 THEN M = FCN
22730  IF IJ = 1 THEN M1 = FCN
22740  IF IJ = 2 THEN M2 = FCN
22750  IF IJ = 3 THEN M3 = FCN
22760  RETURN
```

Figure 8.4—Function Evaluation Subroutine

References

Boyce, William E. and Richard C. DiPrima, *Elementary Differential Equations and Boundary Value Problems*, John Wiley and Sons, Inc., New York, N.Y., 1977.

Hildebrand, F. B., *Introduction to Numerical Analysis*, McGraw-Hill, Inc., New York, N.Y., 1974.

Exercises

1. At time $t=0$ there are Q_0 lbs. of a substance dissolved in 50 gallons of water in a barrel. Water is entering the barrel at the rate of 2 gallons/min. which contains ½ lb. of the substance per gallon. The well-mixed solution leaves the barrel at the same rate. What is the amount of the substance in the barrel at $t=6$? Let $Q_0=Q(t_0)=5$ lbs.

   Hint: the differential equation for this system is

$$Q'(t) = 1 - \frac{Q(t)}{25}$$

2. Solve the following first-order ordinary differential equation at $x = 0$, 1, and 2.

$$y' = \frac{2x^2 + y}{6 - 9y - x} \qquad y(0) = 0$$

3. The following differential equation is not presented in standard form. Find the standard form and find $g(x)$ at $x = 1$, 1.5, and 2.

$$xy' + y = 1 - xy \qquad y(1) = 0$$

4. Classify the order of the following differential equations. Which ones could you solve using the programs in this chapter?
   a. $x^2y'' + 3x(x - y) + 2xy' = 0$
   b. $y' + x^5 - 3x^4 + e^x - 3 = xy'$
   c. $y'' = e^{-5x}$
   d. $x^4y - 9x^3y' - 7 = x^6y' + \cos x$
   e. $(x^4 - 3x^2y)dx + x^2dy = xy$

5. The disintegration of a new radioactive isotope, Dittmannium, is proportional to the amount remaining. What is the expression of this phenomenon as a differential equation? If we had 300 milligrams of Dittmannium at $t = 0$, and had 261 milligrams 6 days later, how much would we have in 24 days?

9

Linear Programming

In many cases of design and analysis, one needs to find the values of certain variables that result in the maximum or minimum value of a function. Further, these variables are often constrained in the values which they may take on. Such a problem is called a mathematical programming problem. In this chapter we examine this type of problem when all the functions are linear; it is therefore referred to as linear programming (LP). We will present the general structure of the LP model and briefly discuss the widely used solution procedure called simplex. A BASIC program is included to solve LP problems using the simplex procedure.

9.1 Linear Programming

Linear programming (LP) is one technique among many designated as mathematical programming. The use of the word "programming" implies nothing about the programming of a computer. In this context it refers to a specified, or programmed, solution procedure. For small problems solutions can be found through hand calculation. Larger problems—in fact, most problems—are solved much more efficiently on the computer.

Problem formulations which may be solved via LP, called linear programs, are distinguishable because all functions used are linear functions. If the functions are nonlinear the correct solution procedure is nonlinear programming, which we do not discuss in this book.

Linear programming has an extensive theoretical background. Time and space do not permit us to develop the theoretical foundations. Linear algebra concepts are extremely important to this development and are used extensively in the solution to problems, as you will see in section 9.3. You may find it useful to supplement the material presented here with other sources. There are many fine books which discuss LP, from the easy and practical to the complex and theoretical. A few books which you may find helpful are Taha (1982), McMillan (1975), Hadley (1963), and Bazaraa and Jarvis (1977).

In which specific situations would you want to use linear programming? First of all, LP requires you to reduce a system or specific problem into a set of linear functions. The specification of these functions is discussed in greater detail in the next section. This conversion to a set of functions is itself a difficult task. It is, in fact, the most important part of the solution process. If you find the optimal solution to a LP formulation that is

not accurate, you have nothing. We do not spend much time on this topic, but that is not indicative of its value. All the sources listed previously discuss the formulation of linear programs in greater detail.

The actual situations in which LP has been used are numerous. It is generally concerned with the optimal allocation of scarce resources. Since almost all resources are scarce for one reason or another, LP is quite applicable. It has been used in such varied situations as planning, scheduling, transportation, systems analysis, and systems design. It has applications in any type of industry or organization.

The general linear program is formulated in a very specific manner. It consists of a single objective function. This is a linear function of the variables of interest, called decision variables, that represents the desired objectives of the problem. This objective can be maximized or minimized. For instance, a company may wish to minimize transportation cost, or maximize profit. A shop foreman may want to minimize machine down time. A designer of rocket engines may wish to maximize thrust. All of these would specify the objective function.

The remaining functions used are called constraints. These are functions which place restrictions on the system and the decision variables. The constraints limit the value of the objective function. The designer of the rocket engine may be required to use a particular kind of fuel. Thus, he wants to maximize thrust "subject to" the constraint of using a particular fuel. Restrictions, or constraints, on the shop foreman may be types of parts produced, number of machine set-ups required, training of employees, or types of machines. Thus, the entire LP formulation consists of a single objective function and a series of constraints.

In the following section, we present the general structure of a linear program including graphical solutions and an example problem. In Section 9.3 we discuss solution procedures for this type of problem, including the simplex procedure. Finally, a BASIC program for solving LP problems is presented in Section 9.4. This includes an example problem and complete explanation of the output provided.

9.2 Basic Formulation

In the previous section we discussed the general structure of a linear programming formulation; a single objective function and a series of constraints. All functions used in the formulation must be linear. There is a basic formulation that represents linear programs. The following basic formulation is offered:

$$\text{Maximize } Z = \sum_{j=1}^{n} c_j x_j \tag{1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \ 2, \ \cdots, \ m \tag{2}$$

$$x_j \geq 0 \qquad j = 1, \ 2, \ \cdots, \ n \tag{3}$$

where

x_j = decision variables; variables of interest

c_j = coefficients of the decision variables in the objective function

a_{ij} = coefficient of the jth decision variable in the ith constraint

b_i = maximum available amount of the ith resource

m = number of constraint functions

n = number of decision variables

This formulation represents a "maximization" problem with "less than or equal to" constraints. Notice from (3) that all the decision variables are required to be nonnegative.

We could also have a "minimization" problem or "greater than or equal to" constraints. If we had both the formulation may look like this:

$$\text{Minimize } Z = \sum_{j=1}^{n} c_j x_j \tag{4}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \qquad i = 1, 2, \cdots, m \tag{5}$$

$$x_j \geq 0 \qquad j = 1, 2, \cdots, n \tag{6}$$

As you know, working with inequalities can be significantly more troublesome than working with equalities. Thus, in order to solve LP problems analytically, we convert the inequality constraints to equality constraints. This procedure is explained thoroughly in Section 9.3.

Graphical Solution

For problems with two decision variables, the functions can all be plotted in two dimensions. We can use this to solve LP problems graphically. Consider the following problem:

$$\text{Maximize } Z = 2x_1 + 5x_2$$

subject to

$$x_1 \leq 4$$

$$x_2 \leq 3$$

$$x_1 + 2x_2 \leq 8$$

$$x_1 \geq 0, \; x_2 \geq 0$$

We can graph the constraints to find a "feasible region." Only decision variables that lie within this region are possible solutions to the problem. Points lying outside the region constitute "infeasible" solutions. The feasible region for this problem is illustrated in Figure 9.1. The dashed line in Figure 9.1 is the objective function with a value of ten; that is $2x_1 + 5x_2 = 10$.

Any point (x_1, x_2) along $2x_1 + 5x_2 = 10$, of course, yields an objective value of 10. From the graph you can see that many points in the feasible region satisfy this. We want to find the point (x_1, x_2) in the feasible region that yields the largest value of the objective function. This will be the "optimal solution."

As we move the objective function up and to the right in Figure 9.1, its value increases. We want to move the objective function to the point where it just touches the feasible region. If we do this for our problem, the result is presented in Figure 9.2.

Notice that the optimal solution occurs at only one point: $x_1 = 2, x_2 = 3$. The resulting objective value is $z = 2(2) + 5(3) = 19$. The optimal point lies at one of the corners of the feasible region. With linear programming, this will always be the case. That is, the optimal solution will always be at a corner point of the feasible region. (In certain cases optimal solutions may also lie along one of the constraints, yielding an infinite number of optimal solutions. Consult the references for further discussion of this phenomenon.) This interesting fact allows us to solve LP problems much more simply. In our problem, since there are five corner points in the feasible region, we need only look

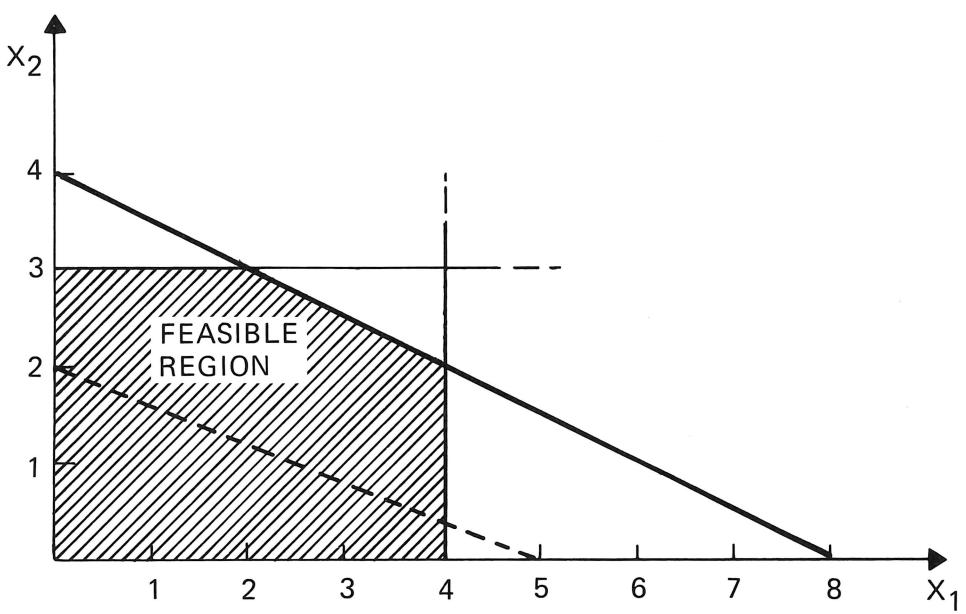


Figure 9.1—Feasible Region for the Example Problem

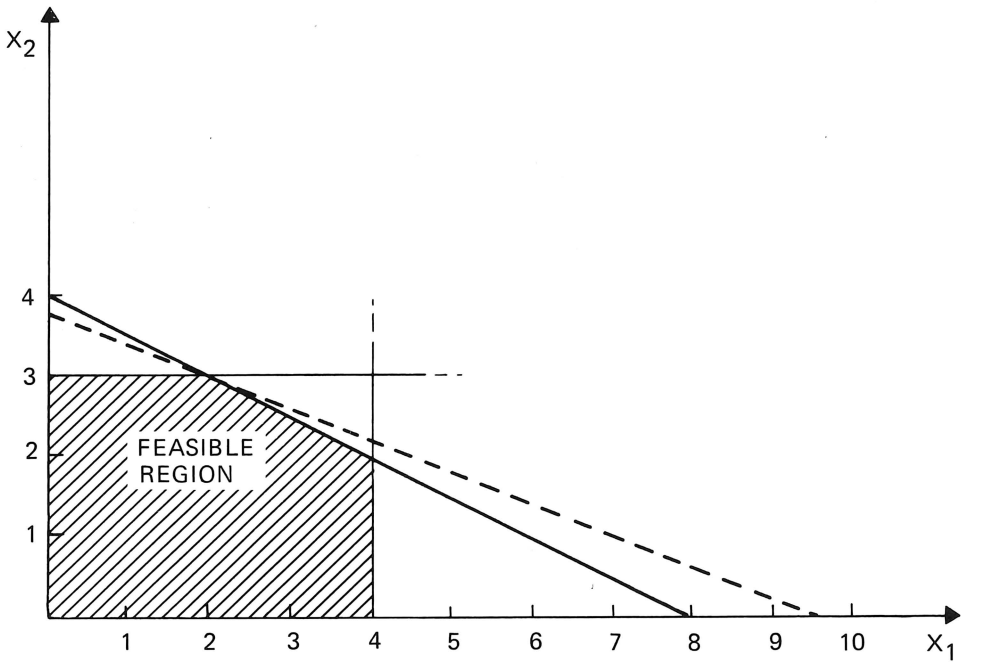


Figure 9.2—Optimal Solution to the Example Problem

at five points to find the optimal solution. For small problems, we could use total enumeration!

You can see that graphical solutions are only applicable for small problems (two decision variables). However, the same principles apply for any size problem. That is, the optimal solution will lie at an extreme point, or corner point, of the feasible region. This fact is used to help find the solution, employing techniques from linear algebra. We present this solution procedure later in the chapter.

If you can formulate a problem so that it fits one of the basic forms above (other forms are also possible; see any of the references listed previously), you can solve it using LP. Getting it into the proper form, that is, formulating the problem as a linear program, is not always an easy task. It takes extensive practice and experience to become confident and comfortable with the formulation process. Below we present a classic linear programming problem. It is used to illustrate the formulation process. Lack of space prevents us from examining more examples of this type. If you find LP useful, we encourage you to review the references at the end of the chapter and study additional formulation exercises.

Linear Programming Example

The following is the classic "peanut problem" adapted from Charnes, Cooper, and Henderson (1953): A manufacturer wishes to determine an optimal program for mixing three grades of nuts consisting of cashews, hazelnuts, and peanuts according to the

Table 1

Mixture	Specifications	Selling Price: c/lb.
A	Not less than 50% cashews Not more than 25% peanuts	40
B	Not less than 25% cashews Not more than 50% peanuts	30
G	No specifications	25

specifications and prices listed in Table 1. Hazelnuts may be introduced into the mixture in any quantity, provided the specifications are met.

If the symbols C for cashews, H for hazelnuts, and P for peanuts are adopted, the restrictions on marketable mixes may be stated in mathematical form:

$$A_C \geq \tfrac{1}{2}A$$

$$A_P \leq \tfrac{1}{4}A$$

$$B_c \geq \tfrac{1}{4}B$$

$$B_P \leq \tfrac{1}{2}B$$

where

$$A_C + A_H + A_P = A$$
$$B_C + B_H + B_P = B$$

The first four inequations state the restrictions on mixtures A and B. Whatever the quantity of A produced, the quantity of cashews going into this mixture, A_C, must account for at least half the total; peanuts going in, A_P, must not exceed one-fourth of the mixture. Similar restrictions apply to B, but no restrictions apply to G. Since mixture G enters only as a residual, it finds no explicit place in these statements of restrictions. Hazelnuts do not appear explicitly in the first group of inequations since no restrictions apply to these nuts. They may appear in any quantity provided that the mixtures meet specifications on cashews and peanuts.

The second group of equations represents definitional relations: The weight of the inputs must equal the weight of the outputs. Some of the quantities may, of course, be zero when the final solution is determined, but the total output of A, say, must be composed of the cashews, A_C, peanuts, A_P, and hazelnuts, A_H, that enter into its makeup.

Substituting these definitional relations into the right-hand side of the inequations, and multiplying by -1, where necessary, to point all inequality signs in the same direction the set may be simplified to:

$$-\tfrac{1}{2}A_C + \tfrac{1}{2}A_P + \tfrac{1}{2}A_H \leq 0$$
$$-\tfrac{1}{4}A_C + \tfrac{3}{4}A_P - \tfrac{1}{4}A_H \leq 0$$
$$-\tfrac{3}{4}B_C + \tfrac{1}{4}B_P + \tfrac{1}{4}B_H \leq 0$$
$$-\tfrac{1}{2}B_C + \tfrac{1}{2}B_P - \tfrac{1}{2}B_H \leq 0$$

Now, suppose that the manufacturer has certain capacity limits on the amounts of inputs that can be employed. Let these limitations and the price of the inputs appear as in Table 2.

These terms need to be interpreted with care before giving them mathematical expression. If "capacity" is interpreted as a maximal limitation on each type of nut, the conditions will be expressed as

$$C = A_C + B_C + G_C \leq 100$$

$$P = A_P + B_P + G_P \leq 100$$

$$H = A_H + B_H + G_H \leq 60$$

In other words, the total amount of cashews, C, going into A, B, and G cannot exceed 100 lb.; the same limitation applies to peanuts; and the total amounts of hazelnuts cannot exceed 60 lb.

There are then 9 unknowns to be determined from the 7 constraints and the nonnegativity constraints. Such a set of relations will, in general, yield an infinite number of solutions. From this infinite set of solutions it is desired to select the best variables according to some criterion or set of criteria. Such a criterion function is found in this problem by constructing the net receipts function. Referring to the set of prices listed in Tables 1 and 2, the following (linear) function may be constructed:

$$Z = 40(A_C + A_P + A_H) + 30(B_C + B_P + B_H) + 25(G_C + G_P + G_H)$$
$$- 65(A_C + B_c + G_C) - 25(A_P + B_P + G_P) - 35(A_H + B_H + G_H)$$

or

$$Z = -25A_C + 15A_P + 5A_H - 35B_C + 5B_P - 5B_H - 40G_C + 0G_P - 10G_H$$

Table 2

Inputs	Capacity: lb./day	Price: c/lb.
C	100	65
P	100	25
H	60	35
Total	260	

9.3 Solution Methods

Solution Via Linear Algebra

In order to solve a LP problem using linear algebra, we must first put the problem in standard form. This standard form must contain an objective function subject to a set of *equality* constraints. That is, each constraint function is equal to the resource available.

Our standard form will have a maximization objective. If the problem dictates a minimization objective, simply multiply the objective function by -1 to convert it to maximization. The standard form is:

$$\text{Maximize } Z = \sum_{j=1}^{n} c_j x_j \tag{7}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = b, \qquad i = 1, 2, \ldots, m \tag{8}$$

$$x_j \geq 0 \qquad j = 1, 2, \ldots, n \tag{9}$$

The x_j may not all be decision variables. Some variables are required to convert constraints from inequalities to equalities. These are called "slack" variables. We would modify our example problem to fit the standard form as follows.

$$\text{Maximize } Z = 2x_1 + 5x_2$$

subject to

$$\begin{aligned} x_1 + x_3 &= 4 \\ x_2 + x_4 &= 3 \\ x_1 + 2x_2 + x_5 &= 8 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned}$$

Notice that variables x_3, x_4, and x_5 do not appear in the objective function (in other words, they have coefficients of zero). These variables merely take up any slack if the left hand side of an original inequality does not equal the right hand side.

The above procedure is used to convert less than or equal to constraints to equality constraints. Similarly, we can convert greater than or equal to constraints to equalities by subtracting "slack" variables. The references listed discuss this in detail. The constraints specifying nonnegativity will be implicit in the solution procedure.

With equality constraints, linear algebra can be used. Remember that the optimal solution will be at an extreme point of the feasible region. This is what is called a "basic" solution. A basic solution is found by assigning some variables the value of zero and solving for the remaining variables. For a problem with n variables and m equality constraints ($n \geq m$), a basic solution is found by setting n-m variables equal to zero (called nonbasic variables) and solving for the remaining m variables (called basic variables). Each basic solution corresponds to the intersection of m constraints in m-space. Some of these intersections will lie in the feasible region and will therefore be feasible solutions. Thus, each basic feasible solution represents a "corner" point of the feasible region. Intersections outside the feasible region yield infeasible solutions. The way we determine whether a solution is feasible or not is by examining the signs of the resulting basic variables. If all are nonnegative, the solution is feasible. If one or more variables are negative, the solution is infeasible. We are, of course, only interested in feasible solutions.

To solve our example problem, we could examine all basic solutions and choose that one which gives the highest value of the objective function. There are a total of eight basic solutions to our problem. We find two of these below. Finding a basic solution consists of solving a set of simultaneous equations. This procedure is not discussed here but is presented in detail in Chapter 5.

Basic solution #1

Nonbasic variables: x_4, x_5

Basic variables: x_1, x_2, x_3

Constraint equations:

$$x_1 + x_3 = 4$$

$$x_2 = 3$$

$$x_1 + 2x_2 = 8$$

Solution: $x_1 = 2 \quad x_2 = 3 \quad x_3 = 2$

Objective value: $Z = 19$

Basic solution #2

Nonbasic variables: x_1, x_4

Basic variables: x_2, x_3, x_5

Constraint equations:

$$x_3 = 4$$

$$x_2 = 3$$

$$2x_2 + x_5 = 8$$

Solution: $x_2 = 3 \quad x_3 = 4 \quad x_5 = 2$

Objective value: $Z = 15$

If we examined all basic solutions, we would find that the optimal solution is the first basic solution presented above (compare this with the graphical solution we found earlier).

As it turns out, we do not have to examine all the basic solutions to find the optimal solution to a LP problem. In the next section we briefly describe the procedure commonly used to solve LP problems. It utilizes the concepts developed so far in an efficient manner to reduce computational effort.

Simplex Procedure

The most commonly used procedure for solving linear programming problems is the simplex method. Developed by George Dantzig in the late 1940s, this is a simple technique which can solve linear programming problems in an efficient manner. The basic concept of the simplex method is easy to understand. We said earlier that the optimal solution to any linear programming problem always occurs at a corner point of the feasible region. All we have to do is find that particular corner point or points that yield the optimal value of the objective function. For large problems this can amount to many possible solution points that must be investigated. The simplex method provides an efficient pattern in which to investigate these points.

The simplex method proceeds from one corner point of the feasible region to another "adjacent" corner point. If the point reached yields the optimal solution, the procedure

stops. If the point is not optimal, the solution moves to the next corner point. The process continues in this fashion until the optimal solution is found. An important fact to note is that the simplex method can detect the optimality of a particular solution without examining any other solutions. This is important because once the optimal solution is found, it is not necessary to examine any additional points. We do not have to evaluate all corner points, but can stop when the optimal solution is found.

Let us refer back to the move from one corner point to an "adjacent" one. What does this mean? Recall the concept of basic variables from the previous section. A particular set of basic variables, a basic solution, corresponds to a particular corner point. To move from one corner point to another, the set of basic variables must change; at least one basic variable must be different. We move to an adjacent corner point by changing only one of the basic variables. Thus, the simplex method moves from one corner point to an adjacent one by making one of the basic variables a nonbasic variable, and making one of the nonbasic variables a basic variable.

The simplex method must also tell us which basic variable to make nonbasic, and which nonbasic variable should become basic. This decision is made according to its effect on the objective function and the feasibility of the solution. The nonbasic variable that "enters" the solution (becomes basic) is that nonbasic variable which will improve the objective function the most, per unit increase in that variable. This is the first decision that is made. The basic variable that "leaves" the solution (becomes nonbasic) is dependent on the variable that enters. The leaving variable is chosen so that the solution remains feasible. In this manner the simplex method guarantees that, starting with a basic solution in the feasible region, each successive basic solution will also be in the feasible region. There will be a leaving basic variable that meets this requirement. (Note: two important topics in this regard are "degeneracy" and "unboundedness." The references at the end of this chapter discuss these in detail.)

The simplex method stops when the current basic solution is optimal. The solution is optimal when none of the nonbasic variables, if they were entered into the solution, would improve the objective function.

A discussion of the actual operation of the simplex method is not possible in this book. You may want to consult the references to learn the mechanics of the procedure and the many special topics which could not be included here. The use of the simplex program is discussed in the following section.

9.4 Using the Simplex Program

Figure 9.3 presents a stand-alone program of the simplex method to solve linear programming problems. In this section we will discuss its use and evaluation of its output.

Program Input

Input to the program consists of the following:

1. Number of variables
2. Number of constraints
3. Identification numbers for variables
4. Coefficients of variables in the objective function

```
1000  REM      THIS PROGRAM USES THE SIMPLEX PROCEDURE TO SOLVE LINEAR
1010  REM      PROGRAMMING PROBLEMS.  THE USER MUST FORMULATE THE PROBLEM
1020  REM      AS A MAXIMIZATION PROBLEM WITH EQUALITY CONSTRAINTS.  THE
1030  REM      INITIAL BASIC VARIABLES MUST BE INCLUDED IN THE FORMULATION.
1040  REM      INPUT MUST INCLUDE THE NUMBER OF CONSTRAINTS, THE NUMBER OF
1050  REM      VARIABLES (INCLUDING THE INITIAL BASIC VARIABLES), THE
1060  REM      COEFFICIENTS OF ALL VARIABLES IN THE OBJECTIVE FUNCTION,
1070  REM      AND THE COEFFICIENTS OF ALL VARIABLES IN THE
1080  REM      CONSTRAINTS WITH THE EXCEPTION OF INITIAL BASIC VARIABLES,
1090  REM      WHOSE COEFFICIENTS ARE AUTOMATICALLY SUPPLIED BY THE
1100  REM      PROGRAM.
1110  REM
1120 DOS$ =  CHR$ (4): REM    CTRL - D
1130 IT = 0
1140 EP = .0000005
1150  INPUT "TABLEAUS IN FORTY OR EIGHTY COLUMN FORMAT (F/E)?";S$
1160  REM    SET COLUMN PRINTING PARAMETERS
1170  IF S$ = "F" THEN T% = 3:U% = 8
1180  IF S$ = "E" THEN T% = 8:U% = 12
1190  PRINT "ENTER THE FOLLOWING PARAMETERS"
1200  PRINT : INPUT "NUMBER OF CONSTRAINT EQUATIONS, NC ? ";NC
1210  PRINT
1220  INPUT "NUMBER OF VARIABLES, INCLUDING INITIAL  BASIS ? ";NV: PRINT
1230  INPUT "ENTER 1 -- ALL TABLEAUS : 2--ONLY FIRST AND LAST TABLEAUS ? ";IR: PRINT
1240  INPUT "ENTER THE TITLE OF THE PROBLEM ? ";A$
1250  PRINT : PRINT
1260  DIM A(NC,NV),P(NC),JV(NV),C(NV),IV(NV),CC(NV),CI(NV)
1270  DIM ZJ(NV),CM(NV)
1280  PRINT "ENTER THE VARIABLE IDENTIFICATION"
1290  PRINT "NUMBERS BEGINNING WITH THE"
1300  PRINT "INITIAL BASIC VARIABLES"
1310  FOR J = 1 TO NV
1320  INPUT JV(J)
1330  NEXT J
1340  PRINT : PRINT
1350  PRINT "ENTER THE OBJECTIVE FUNCTION"
1360  PRINT "COEFFICIENTS OF THE VARIABLES YOU HAVE"
1370  PRINT "JUST LISTED"
1380  FOR J = 1 TO NV
1390  IF J > NC THEN  GOTO 1440
1400 JJ = NV - NC + J
1410  PRINT : PRINT "ENTER THE COST FOR VARIABLE ";JJ;
1420  INPUT C(J)
1430  GOTO 1470
1440 J2 = J - NC
1450  PRINT : PRINT "ENTER THE COST FOR VARIABLE ";J2;
1460  INPUT C(J)
1470  NEXT J
1480 ND = NC + 1
1490  PRINT : PRINT "FOR EACH CONSTRAINT EQUATION, ENTER"
1500  PRINT : PRINT "FIRST THE VALUE OF THE RIGHT HAND SIDE"
1510  PRINT : PRINT "AND THEN ENTER THE COEFFICIENTS OF ALL"
1520  PRINT : PRINT "VARIABLES IN THAT CONSTRAINT. (NOTICE"
1530  PRINT : PRINT "THAT A VARIABLE NOT APPEARING IN A"
1540  PRINT : PRINT "PARTICULAR CONSTRAINT HAS A COEFFICIENT"
1550  PRINT : PRINT "OF ZERO). DO NOT ENTER COEFFICIENTS FOR"
1560  PRINT : PRINT "ANY VARIABLES CLASSIFIED AS ORIGINAL"
1570  PRINT : PRINT "BASIC VARIABLES."
1580  PRINT : PRINT
1590  FOR I = 1 TO NC
1600  PRINT
1610  PRINT "THE FOLLOWING QUESTIONS PERTAIN TO      CONSTRAINT ";I
1620  PRINT
1630  INPUT "ENTER THE VALUE OF THE RIGHT HAND SIDE?";P(I)
1640  FOR J = ND TO NV
```

```
1650  PRINT "INPUT THE COEFFICIENT OF VARIABLE ";J - NC;
1660  INPUT A(I,J)
1670  NEXT J
1680  NEXT I
1690  FOR I = 1 TO NC
1700  FOR J = 1 TO NC
1710 A(I,J) = 0: REM    BEGIN SETTING COEFFICIENTS FOR BASIC VARIABLES
1720  NEXT J
1730 CI(I) = C(I): REM    SAVE INITIAL COST COEFFICIENTS
1740 IV(I) = JV(I)
1750 A(I,I) = 1: REM    SET IDENTITY MATRIX
1760  NEXT I
1770  PRINT : INPUT "WOULD YOU LIKE TO CHANGE INPUT DATA(Y/N) ? ";C$
1780  IF C$ = "Y" THEN  GOSUB 3040
1790  PRINT : INPUT "OUTPUT TO CRT OR PRINTER (C/P) ? ";Q$
1800  PRINT
1810  IF Q$ = "P" THEN  INPUT "TURN ON PRINTER, HIT RETURN WHEN READY ";S$
1820  IF Q$ = "P" THEN  PRINT DOS$;"PR#1"
1830  PRINT "SOLUTION TO ";A$
1840 CZ =  - 8.999999E + 19
1850  FOR J = 1 TO NV
1860 Z = 0
1870  FOR I = 1 TO NC
1880 Z = Z + A(I,J) * CI(I)
1890  NEXT I
1900 TN = C(J) - Z
1910 ZJ(J) = Z
1920 CM(J) = TN
1930  IF CZ > TN THEN  GOTO 1960
1940 CZ = TN
1950 JI = J
1960  NEXT J
1970 IT = IT + 1
1980  IF IR <  > 2 THEN  GOTO 2010
1990  IF IT = 1 THEN  GOTO 2010
2000  IF CZ >  = EP THEN  GOTO 2720
2010 VL = 0
2020  FOR I = 1 TO NC
2030 VL = VL + P(I) * CI(I)
2040  NEXT I
2050  PRINT : PRINT "THE FOLLOWING RESULTS ARE FOR ITERATION NUMBER ";IT
2060  PRINT : PRINT "OBJ. COEF.";
2070  FOR J = 1 TO NV
2080 JJ = J + NC
2090  IF JJ > NV THEN JJ = JJ - NV
2100  IF J <  = T% THEN  GOTO 2150
2110 ISP =  INT ((J - 1) / T%)
2120  IF J = (ISP * T%) + 1 THEN  PRINT
2130  POKE 36,(7 + (J - (ISP * T%)) * 8)
2140  GOTO 2160
2150  POKE 36,(7 + J * 8)
2160  PRINT  INT (C(JJ) * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5);
2170  NEXT J
2180  PRINT : PRINT : PRINT "VARIABLES";
2190  FOR J = 1 TO NV
2200 JJ = J + NC
2210  IF JJ > NV THEN JJ = JJ - NV
2220  IF J <  = T% THEN  GOTO 2270
2230 ISP =  INT ((J - 1) / T%)
2240  IF J = (ISP * T%) + 1 THEN  PRINT
2250  POKE 36,(7 + (J - (ISP * T%)) * 8)
2260  GOTO 2280
2270  POKE 36,(7 + J * 8)
2280  PRINT  INT (JV(JJ) * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5);
2290  NEXT J
```

```
2300  PRINT :
2310  FOR I = 1 TO NC
2320  PRINT
2330  PRINT IV(I);
2340  HTAB 4
2350 ZZ =  INT (P(I) * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
2360  PRINT ZZ;
2370  FOR J = 1 TO NV
2380 JJ = J + NC
2390  IF JJ > NV THEN JJ = JJ - NV
2400  IF J <  = T% THEN  GOTO 2450
2410 ISP =  INT ((J - 1) / T%)
2420  IF J = (ISP * T%) + 1 THEN  PRINT
2430  POKE 36,(7 + (J - (ISP * T%)) * 8)
2440  GOTO 2460
2450  POKE 36,(7 + J * 8)
2460 ZZ =  INT (A(I,JJ) * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
2470  PRINT ZZ;
2480  NEXT J
2490  PRINT :
2500  NEXT I
2510  PRINT : PRINT
2520  PRINT "RCST";
2530  HTAB 6
2540 ZZ =  INT (VL * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
2550  PRINT ZZ;
2560  FOR J = 1 TO NV
2570 JJ = J + NC
2580  IF JJ > NV THEN JJ = JJ - NV
2590  IF J <  = T% THEN  GOTO 2640
2600 ISP =  INT ((J - 1) / T%)
2610  IF J = (ISP * T%) + 1 THEN  PRINT
2620  POKE 36,(7 + (J - (ISP * T%)) * 8)
2630  GOTO 2650
2640  POKE 36,(7 + J * 8)
2650 ZZ =  INT (CM(JJ) * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
2660  PRINT ZZ;
2670  NEXT J
2680  PRINT :
2690  IF CZ < EP THEN  GOTO 3020
2700  IF Q$ = "P" THEN  GOTO 2720
2710  PRINT : INPUT "PRESS RETURN TO CONTINUE SOLUTION -> ";B$
2720 TT = 9 * 10 ^ 19
2730  FOR I = 1 TO NC
2740  IF A(I,JI) <  = EP THEN  GOTO 2790
2750 RT = P(I) / A(I,JI)
2760  IF TT < RT THEN  GOTO 2790
2770 TT = RT
2780 IL = I
2790  NEXT I
2800  IF TT < 900000000000000 THEN  GOTO 2830
2810  PRINT "THE OBJECTIVE FUNCTION IS UNBOUNDED"
2820  GOTO 3020
2830 PV = A(IL,JI)
2840 P(IL) = P(IL) / PV: REM             CALCULATE THE NEW RIGHT HAND SIDE
2850  FOR J = 1 TO NV
2860 A(IL,J) = A(IL,J) / PV: REM         CALCULATE THE NEW PIVOT ROW
2870  NEXT J
2880  FOR I = 1 TO NC
2890  IF I = IL THEN  GOTO 2960
2900 P(I) = P(I) - P(IL) * A(I,JI): REM    NEW RHS FOR OTHER BASIC VARIABLES
2910  FOR J = 1 TO NV
2920  IF J = JI THEN  GOTO 2940
2930 A(I,J) = A(I,J) - A(I,JI) * A(IL,J): REM    FIND THE BASIC VARIABLE ROW ELEMENTS
2940  NEXT J
```

```
2950 A(I,JI) = 0: REM                    CREATE A UNIT VECTOR COLUMN
2960  NEXT I
2970 EP = EP + .0000005
2980 CI(IL) = C(JI): REM                 ASSIGN NEW COSTS TO THE BASIC C VECTOR
2990 IV(IL) = JV(JI): REM                SWITCH OTHER VARIABLE ASSIGNMENTS
3000 A(IL,JI) = 1: REM                   TO COMPLETE THE UNIT VECTOR COLUMN
3010  GOTO 1840
3020  PRINT DOS$;"PR#0"
3030  END
3040  REM  ************************************************************
3050  REM   SUBROUTINE FOR EDITING INPUT DATA
3060  REM
3070  PRINT : PRINT "ENTER THE NUMBER FOR DESIRED CHANGE": PRINT
3080  PRINT "    1: EXIT THE PROGRAM"
3090  PRINT "    2: CHANGE OBJECTIVE COEFFICIENTS"
3100  PRINT "    3: CHANGE CONSTRAINT EQUATION"
3110  PRINT "       COEFFICIENTS"
3120  PRINT "    4: CHANGE RIGHT HAND SIDE VALUES"
3130  PRINT "    5: EXIT EDIT ROUTINE"
3140  PRINT : INPUT "OPTION ? ";ED
3150  IF ED = 5 THEN  GOTO 4000
3160  IF ED = 1 THEN  END
3170  IF ED = 2 THEN  GOTO 3380
3180  IF ED = 3 THEN  GOTO 3640
3190  PRINT "CURRENT RIGHT HAND VALUES ARE :"
3200  FOR I = 1 TO NC
3210  PRINT "FOR CONSTRAINT ";I;" RHS IS ";P(I)
3220  NEXT I
3230  PRINT : INPUT "DISPLAY CONSTRAINT LIST AGAIN (Y/N) ? ";C$
3240  IF C$ = "Y" THEN  GOTO 3190
3250  PRINT : INPUT "ENTER CONSTRAINT NUMBER FOR CHANGE ? ";DCH
3260  PRINT : INPUT "ENTER NEW RIGHT HAND SIDE VALUE ? ";P(DCH)
3270  PRINT : PRINT "ENTER :"
3280  PRINT : PRINT "   M: MODIFY ANOTHER RIGHT HAND SIDE"
3290  PRINT : PRINT "   R: RETURN TO EDIT MENU"
3300  PRINT : PRINT "   E: EXIT EDIT ROUTINE"
3310  PRINT : INPUT T$
3320  IF T$ = "M" THEN  GOTO 3230
3330  IF T$ = "R" THEN  GOTO 3070
3340  IF T$ = "E" THEN  GOTO 4000
3350  REM    FLAG INPUT ERROR AND RE-ENTER
3360  PRINT "INPUT ERROR: RE-ENTER"
3370  GOTO 3270
3380  REM    FOR MODIFYING OBJECTIVE COEFFICIENTS
3390  PRINT : PRINT "CURRENT OBJECTIVE COEFFICIENTS :"
3400  FOR I = 1 TO NV
3410 J = I + NC
3420  IF I > NV - NC THEN J = I - NV + NC
3430  PRINT : PRINT "FOR VARIABLE ";I;" COEFFICIENT IS ";C(J)
3440  NEXT I
3450  PRINT : INPUT "DISPLAY COEFFICIENT LIST AGAIN (Y/N) ?";C$
3460  IF C$ = "Y" THEN  GOTO 3390
3470  PRINT : INPUT "ENTER VARIABLE NUMBER FOR CHANGE ? ";TJJ
3480  IF TJJ > NV - NC THEN  GOTO 3510
3490 TJJ = TJJ + NC
3500  GOTO 3520
3510 TJJ = TJJ - NV + NC
3520  PRINT : INPUT "ENTER NEW COEFFICIENT VALUE ? ";C(TJJ)
3530  PRINT : PRINT "ENTER :"
3540  PRINT : PRINT "   M: MODIFY ANOTHER OBJECTIVE COEFF."
3550  PRINT : PRINT "   R: RETURN TO EDIT MENU"
3560  PRINT : PRINT "   E: EXIT EDIT ROUTINE"
3570  PRINT : INPUT T$
3580  IF T$ = "M" THEN  GOTO 3450
3590  IF T$ = "R" THEN  GOTO 3070
3600  IF T$ = "E" THEN  GOTO 4000
```

```
3610  REM   FLAG INPUT ERROR AND RE-ENTER
3620  PRINT : PRINT "INPUT ERROR: RE-ENTER"
3630  GOTO 3530
3640  REM    FOR MODIFYING CONSTRAINT COEFFICIENTS
3650  PRINT : PRINT "BASIC VARIABLE WILL NOT BE CONSIDERED"
3660  PRINT : INPUT "DISPLAY ALL CONSTRAINT COEFFICIENTS    (Y/N) ? ";C$
3670  IF C$ = "N" THEN  GOTO 3760
3680  FOR I = 1 TO NC
3690  PRINT
3700  PRINT "FOR CONSTRAINT ";I;" COEFFICIENTS ARE :"
3710  FOR J = NC + 1 TO NV
3720  PRINT "VAR ";J - NC;" = ";A(I,J);"  ";
3730  NEXT J
3740  NEXT I
3750  PRINT
3760  INPUT "DISPLAY COEFFICIENTS FOR A PARTICULAR   CONSTRAINT (Y/N) ? ";C$
3770  IF C$ = "N" THEN  GOTO 3850
3780  PRINT : INPUT "CONSTRAINT NUMBER ? ";B
3790  FOR J = NC + 1 TO NV
3800  PRINT : PRINT "VAR ";J - NC;" = ";A(B,J);
3810  NEXT J
3820  PRINT
3830  INPUT "DISPLAY ANOTHER CONSTRAINT (Y/N) ? ";C$
3840  IF C$ = "Y" THEN  GOTO 3780
3850  PRINT : INPUT "ENTER CONSTRAINT NUMBER FOR CHANGE ? ";CCH
3860  PRINT : INPUT "ENTER VARIABLE NUMBER FOR CHANGE ? ";DCH
3870 DCH = DCH + NC
3880  PRINT : INPUT "ENTER NEW COEFFICIENT VALUE ? ";A(CCH,DCH)
3890  PRINT : PRINT "ENTER :"
3900  PRINT : PRINT "   M: MODIFY ANOTHER CONSTRAINT COEFF."
3910  PRINT : PRINT "   R: RETURN TO EDIT MENU"
3920  PRINT : PRINT "   E: EXIT EDIT ROUTINE
3930  PRINT : INPUT T$
3940  IF T$ = "M" THEN  GOTO 3650
3950  IF T$ = "R" THEN  GOTO 3070
3960  IF T$ = "E" THEN  GOTO 4000
3970  REM    FLAG INPUT ERROR AND RE-ENTER
3980  PRINT : PRINT "INPUT ERROR: RE-ENTER"
3990  GOTO 3890
4000  RETURN
```

Figure 9.3—Simplex Program

5. Coefficients of variables in the constraints
6. Right hand side value of each constraint

The program makes four major assumptions regarding your preparation of the problem for solution.

1. The objective function is to be maximized. If your objective function is to be minimized, simply multiply the function by -1 to convert it to a maximization problem.
2. All constraints have been converted to equality constraints.
3. All variables are restricted to non-negative values.
4. All right hand side values are non-negative.

Once the problem is in this format, you only need an initial solution to start the program. The stages of the solution are presented in a table called a "tableau." This will be discussed in detail in the output section.

The program requires an initial basic solution. This generally consists of slack variables. There will be a basic variable for each constraint. An initial basic variable must be one which appears in only one constraint and has a coefficient of +1 (if it is different from +1, you may be able to make it +1 by dividing the entire constraint equation by a constant). This makes slack variables a natural. Consider an earlier problem.

Maximize $Z = 2x_1 + 5x_2$

subject to

$$x_1 \leq 4$$

$$x_2 \leq 3$$

$$x_1 + 2x_2 \leq 8$$

$$x_1, x_2 \geq 0$$

With slack variables to create equality constraints:

Maximize $Z = 2x_1 + 5x_2$

subject to

$$x_1 + x_3 = 4$$

$$x_2 + x_4 = 3$$

$$x_1 + 2x_2 + x_5 = 8$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0$$

In this problem, the slack variables x_3, x_4, and x_5 would be initial basic variables since they each appear in only one constraint and have coefficients of +1. However, any variable which meets these conditions can be an initial basic variable. Remember, there must be one from each constraint.

The problem above is in the form required to run the program. Let us look at the program operation as the inputs are requested and use this problem as an example. Below we present the input portion of the program and answer according to this problem.

```
ENTER THE FOLLOWING PARAMETERS
NUMBER OF CONSTRAINT EQUATIONS,NC? 3

NUMBER OF VARIABLES, INCLUDING INITIAL
BASIS? 5

ENTER 1--ALL TABLEAUS : 2--ONLY FIRST
AND LAST TABLEAUS? 1
ENTER THE TITLE OF THE PROBLEM? EXAMPLE
PROBLEM NO 1
```

In the above questions we are defining the problem. We have three constraint equations and five variables (x_1 through x_5). The next question allows you the option of seeing all tableaus or just the first and last. Finally, you will want a title for your problem.

```
ENTER THE VARIABLE IDENTIFICATION
NUMBERS BEGINNING WITH THE
INITIAL BASIC VARIABLES
?3
?4
?5
?1
?2
```

These are the last problem definition questions prior to inputting the actual problem numbers. We must be able to identify each variable with a number. The easiest to use are the subscript numbers as we have done here. Thus, 3 is x_3, 4 is x_4, etc. Notice that the initial basic variables are entered first. Their order must match the constraints. That is, x_3 is in constraint 1, x_4 is in constraint 2, and x_5 is in constraint 3. Order does not matter for the other variables.

```
ENTER THE OBJECTIVE FUNCTION
COEFFICIENTS OF THE VARIABLES YOU HAVE
JUST LISTED
ENTER THE COST FOR VARIABLE 3 ? 0
ENTER THE COST FOR VARIABLE 4 ? 0
ENTER THE COST FOR VARIABLE 5 ? 0
ENTER THE COST FOR VARIABLE 1 ? 2
ENTER THE COST FOR VARIABLE 2 ? 5
```

Here we input the objective function coefficients of each variable. If a variable does not appear in the objective function, its coefficient is zero, as is the case for x_3, x_4, and x_5.

```
FOR EACH CONSTRAINT EQUATION, ENTER
FIRST THE VALUE OF THE RIGHT HAND SIDE
AND THEN ENTER THE COEFFICIENTS OF ALL
VARIABLES IN THAT CONSTRAINT. (NOTICE
THAT A VARIABLE NOT APPEARING IN A
PARTICULAR CONSTRAINT HAS A COEFFICIENT
OF ZERO). DO NOT ENTER COEFFICIENTS FOR
ANY VARIABLES CLASSIFIED AS ORIGINAL
BASIC VARIABLES.
```

This is printed as instruction for the next set of inputs; entering the constraint equations. The questions requesting the specific input provide additional guidance. The coefficients for the original basic variables are automatically provided by the program and should not be entered by the user.

```
THE FOLLOWING QUESTIONS PERTAIN TO
CONSTRAINT 1

ENTER THE VALUE OF THE RIGHT HAND SIDE? 4
INPUT THE COEFFICIENT OF VARIABLE 1 ? 1
INPUT THE COEFFICIENT OF VARIABLE 2 ? 0

THE FOLLOWING QUESTIONS PERTAIN TO
CONSTRAINT 2

ENTER THE VALUE OF THE RIGHT HAND SIDE? 3
INPUT THE COEFFICIENT OF VARIABLE 1 ? 0
INPUT THE COEFFICIENT OF VARIABLE 2 ? 1
```

```
THE FOLLOWING QUESTIONS PERTAIN TO
CONSTRAINT 3

ENTER THE VALUE OF THE RIGHT HAND SIDE? 8
INPUT THE COEFFICIENT OF VARIABLE 1 ? 1
INPUT THE COEFFICIENT OF VARIABLE 2 ? 2
```

This is where you input the constraint equations. Recall that the order of the constraints and the order of the original basic variables must match. The program asks for specific right hand side values and coefficients for particular variables within each constraint.

The input is now complete and the computer can begin to use the simplex method to solve our problem. An editor is included, in the event you wish to change your input prior to solving the problem. You will also have the option of specifying a 40- or 80-column output. The 80-column output was tested on the IIe. It may not operate appropriately with other 80-column boards. In the next section we will examine the output of the program.

There are many additional linear programming formulations and other issues which we have not been able to discuss. In order to fully utilize the program presented you should become familiar with these additional concepts. The references provide a wealth of knowledge which should prove very beneficial to you.

Program Output

The output of the program is somewhat controllable by you. That is, you can request an output at each iteration or only the first and last iterations. An iteration is a move from one corner point to another. At each iteration a "tableau" is printed. This tableau is full of information. Some of this information will be meaningless to you unless you delve into linear programming beyond our brief discussion here. It is provided for those who can put it to use. Since it follows the standard tableau format, it will be easy to understand.

The tableau reveals three very important conditions of the current solution: the basic variables, the solution value of each basic variable, and the non-basic variables which would improve the objective function if they were made basic (this, in turn, tells us when we have reached the optimal solution). Figure 9.4 presents the initial tableau for our example problem (in 40-column format). Let us analyze it.

```
THE FOLLOWING RESULTS ARE FOR ITERATION
NUMBER 1

OBJ. COEF.        2          5                  0
                  0          0

VARIABLES         1          2          3
                  4          5

 3    4           1          0          1
                  0          0

 4    3           0          1          0
                  1          0

 5    8           1          2          0
                  0          1

RCST 0           2          5          0
                  0          0
```

Figure 9.4—Tableau 1

Look first at the second row, labeled "VARIABLES." Each variable in the problem has its own column in the tableau. They are labeled according to the identification numbers you have provided. If there are more than three variables in your problem (eight for the 80-column option), the variables after the third (eighth) one will be shifted down one row. This will result in at least one column being used for at least two variables; one underneath the other. The variables assigned to each column will remain the same throughout the solution procedure. You can see this wraparound effect in the tableaus for this problem.

The first row, labeled "OBJ.COEF.," lists the coefficients in the objective function for the respective variables listed in the second row. These also will not change during problem solution. You should check these to verify that your input was correct.

The left-most column lists the basic variables for the current solution. In Figure 9.4 the basic variables, going down the column, are variables $3(x_3)$, $4(x_4)$, and $5(x_5)$. These will change as the simplex method moves toward the optimal solution.

The column immediately to the right of the first column yields the solution values for the basic variables. In Figure 9.4, our solution is

$$x_3 = 4$$

$$x_4 = 3$$

$$x_5 = 8$$

The last element in this column, the one not associated with a particular basic variable, represents the current value of the objective function. In Figure 9.4, this value is zero.

You may notice that we have not mentioned anything about variables x_1 and x_2. We do not need to. At this iteration they are nonbasic variables and therefore their value is zero.

In summary, Figure 9.4 presents this solution:

$$x_1 = 0 \quad x_2 = 0 \quad x_3 = 4 \quad x_4 = 3 \quad x_5 = 8 \quad Z = 0$$

You should verify that this solution is correct by plugging these values into the problem formulation to see that all constraints are satisfied and the correct objective function value results.

The tableau also tells us whether or not the current solution is the optimal solution. This is found in the row labelled "RCST" (reduced cost). The first element of this row is the value of the objective function; the remaining elements reveal the condition of the solution. Notice that there is a value in this row for every variable in our problem. The values corresponding to basic variables will always be zero. We can see in Figure 9.4 that the values in the reduced cost row are zero in the columns of variables x_3, x_4, and x_5. The values in columns of nonbasic variables indicate the marginal desirability of allowing them to "enter" the solution. If the value is positive, then the entrance of the corresponding nonbasic variable into the solution will improve (increase) the value of the objective function; something we would want to do. If it is negative, then that variable should remain nonbasic. If it is zero, the objective function value will not change; we will leave the variable as nonbasic.

In Figure 9.4 the values in the reduced cost row for variables x_1 and x_2 are both positive, meaning they could both enter the solution. However, the simplex method

chooses only one variable at a time to enter. The general procedure, and that followed by the program, is to choose that variable with the largest value in the reduced cost row. In this case this is x_2. In the next iteration, x_2 will become a basic variable and one of the current basic variables will become nonbasic.

The tableau resulting from this change is presented in Figure 9.5. You will notice that x_2 has replaced x_4 as a basic variable. The solution with these basic variables (x_3, x_2, and x_5) is found in Figure 9.5 as:

$x_1 = 0$ $x_2 = 3$ $x_3 = 4$ $x_4 = 0$ $x_5 = 2$ $Z = 15$

```
THE FOLLOWING RESULTS ARE FOR ITERATION
NUMBER 2


OBJ. COEF.          2          5          0
                    0          0


VARIABLES           1          2          3
                    4          5


3   4               1          0          1
                    0          0


2   3               0          1          0
                    1          0


5   2               1          0          0
                   -2          1


RCST 15             2          0          0
                   -5          0
```

Figure 9.5—Tableau 2

If we examine the reduced cost row we find that the value in the x_1 column, 2, means that the objective function value can be increased if x_1 enters the solution. The value in the x_4 column, -5, means that x_4 should remain nonbasic.

If x_1 enters the solution, we obtain the tableau in Figure 9.6. The resulting solution is:

$x_1 = 2$ $x_2 = 3$ $x_3 = 2$ $x_4 = 0$ $x_5 = 0$ $Z = 19$

```
THE FOLLOWING RESULTS ARE FOR ITERATION
NUMBER 3
```

```
OBJ. COEF.        2          5          0
                  0          0

VARIABLES         1          2          3
                  4          5

3   2             0          0          1
                  2         -1

2   3             0          1          0
                  1          0

1   2             1          0          0
                 -2          1

RCST 15           0          0          0
                 -1         -2
```

Figure 9.6—Tableau 3

This is the optimal solution to the problem. How do we know? Because there are no positive values in the reduced cost row (with the exception of the objective function value). This means that there are no nonbasic variables which, if they were to enter the solution, would increase the value of the objective function. Therefore, the solution in Figure 9.6 yields the largest value of the objective function for all choices of the variables which satisfy the stated constraints.

The option of seeing each iteration tableau is yours. The program will ask if you would like to see all tableaus or only the first and last. You will always want to see the first to verify your input, and the last for the optimal solution.

For formatting purposes, the program specifies a format for the tableau entries. Objective coefficients are limited to 9999.99, constraint coefficients to 9999.99, and right hand side values to 9999999.99 for printing purposes. You will want to make sure that your problem is properly scaled. The current formatting will be sufficient for most problems.

The remaining portions of the tableau are provided for those who spend additional time studying linear programming. For the novice, their purpose is unimportant. After further study, their meaning will become evident. Therefore, a discussion is omitted here.

Finally, if the problem is such that the objective function can increase without bound, the program will inform you of this situation. This occurrence usually implies that your model is not truly reflective of the modeled situation.

Program Coverage

Many special topics and considerations in linear programming were not explicitly discussed here, particularly with respect to the operation of the program. An attempt was made to provide a basic introduction to the concepts of the simplex method and the program application. The program can also be used in a more advanced manner, but the authors feel that a detailed description here is not feasible.

The program can be used in an imaginative way to address advanced problems but is dependent on the user's understanding of the simplex method. For example, the program

can be used with artificial variables in a two-phase procedure. As the user becomes more knowledgable in linear programming, a detailed description of the use of the program for more advanced problems will not be necessary.

9.5 Summary

In this chapter we have presented linear programming. This is a very powerful and popular tool which is applicable in many areas. In these few pages we were only able to touch the surface of the topic.

The "art" of linear programming is encompassed in the formulation of the model. We illustrated this with the "Peanut Problem" of Charnes, Cooper, and Henderson. We encourage you to work on the formulation examples. If the model is incorrect, the solution will be incorrect.

The "science" of linear programming is in finding the optimal solution. We briefly discussed graphical solution concepts and algebraic solution procedures. We also discussed the fundamental concepts behind the simplex method, which was a major breakthrough in the solution of linear programming problems. The computer program in this chapter used the simplex method.

Since only a brief introduction to linear programming is provided here, the interested reader is encouraged to consult the references for more advanced topics.

References

Bazaraa, Mokhtar S. and John J. Jarvis, *Linear Programming and Network Flows*. John Wiley and Sons, Inc., New York, N.Y., 1977.

Charnes, A., W. W. Cooper, and A. Henderson, *An Introduction to Linear Programming*. John Wiley and Sons, Inc., New York, 1953.

Hadley, G., *Linear Programming*. Addison-Wesley Publishing Co., Reading, Mass., 1963.

McMillan, Claude, *Mathematical Programming*. John Wiley and Sons, Inc., New York, N.Y., 1975.

Taha, Hamdy A., *Operations Research*. Macmillan Publishing Company, Inc., New York, N.Y., 1982.

Exercises

1. Solve the following linear programming problem.

   Maximize $z = 8x_1 + 2x_2 + 5x_3$
   subject to

   $$2x_1 + 2x_2 + x_3 \leq 12$$
   $$3x_1 - x_2 + 2x_3 \leq 9$$
   $$x_2 + 3x_3 \leq 5$$
   $$x_1, x_2, x_3 \geq 0$$

2. An oil company requires two different types of crude oil to produce leaded, unleaded, and diesel fuels. A particular profit is made on each type of fuel, and each fuel consists of certain proportions of each crude type. Profit is limited because crude oil is limited. The table below presents the appropriate summary information.

Percentage Required of each Crude type for Fuels

Fuel Type	Crude Type		Profit Per bbl
	1	2	
leaded	0.4	0.6	0.12
unleaded	0.5	0.5	0.09
diesel	0.7	0.3	0.10
Available crude (bbl)	400	300	

Formulate and solve a linear programming problem which maximizes profit while not exceeding crude oil availability.

3. Different crops are affected differently by various fertilizers. Each fertilizer/crop combination results in a particular crop yield. The table below illustrates these yields for a few crops of interest.

Yields for Fertilizer/Crop Combinations
(in bushels/lb. of Fertilizer)

	Crop			Maximum Availability	Cost/
Fertilizer	Corn	Wheat	Soybeans	(lbs.)	lb.
A	15	31	20	100	$20
B	18	25	16	125	25
C	12	35	30	80	20
Selling Price Per Bushel	$2	$1.50	$1.80		

Formulate and solve a linear programming problem to maximize profit from crop sales.

4. A manufacturing firm produces two types of power tools. Tool A requires 30 minutes of casting, 45 minutes of machining and 20 minutes of assembly. Tool B requires 40, 30, and 25 minutes, respectively, of these processes. The firm is limited to 40 hours of each process per week. If output of tool A is twice as valuable as output of tool B, how many units of each should be produced per week?
5. Solve the "Peanut Problem" presented in this chapter.

10

Forecasting with Exponential Smoothing

Forecasting is an important function because a good forecast reduces uncertainty and thereby aids in decision making. Unfortunately, much of forecasting is an art. Some techniques, however, have been developed and refined that assist us in making better forecasts. One of these, linear regression, was discussed in Chapter 4. Regression is almost always used when the forecast model contains a relationship between one or more variables and the dependent variable being forecast. For example, the projected number of new home starts for next month might be a function of the interest rate. In this case, the interest rate might be used to forecast the home starts.

Another forecasting technique called exponential smoothing will be introduced in this chapter. Exponential smoothing may be employed to forecast future values in a time series using only the historical values in the time series to make the projection. No other independent variable, such as interest rate, is involved. The monthly sales of microcomputers by firm ABC is a time series. Exponential smoothing could be used to forecast the future sales of microcomputers considering only the past sales.

Time-series forecasting techniques such as exponential smoothing are primarily used for short term forecasts (forecasts up to a year in the future). One reason is that these procedures extrapolate the historical observations into the future without considering changes to the underlying process, such as changes in economic conditions and demand.

10.1 Single Exponential Smoothing

An N-period moving average is a popular time-series forecasting technique. However, exponential smoothing can provide better results with less effort. An N-period moving average forecast is made by averaging the last N-periods, then using this value to make the forecast. At the end of the next period, the oldest observation is removed from the series and the last observation is added, always keeping the N most recent observations.

$$X_{t+1} = \sum_{i=t-N+1}^{t} X_i/N \tag{1}$$

Each observation is equally weighted, 1/N. Also, N elements of data must be maintained. This procedure works well if the process being modeled is level (horizontal) and not changing over time. Single exponential smoothing could also be used, and we will see that it has some advantages.

An expression for single exponential smoothing is

$$F_{t+1} = \alpha X_t + (1 - \alpha)F_t \tag{2}$$

where F_t = forecast for period t,

X_t = actual observation for period t,

F_{t+1} = forecast for period t+1, and

α = smoothing constant, $0 \leq \alpha \leq 1$.

One advantage of exponential smoothing can be readily seen. Only three data elements are required: the most recent forecast, the most recent actual observation, and the smoothing constant. Because of these small data storage requirements, exponential smoothing is often used in computerized inventory control systems in which thousands of items must be forecast.

Some manipulation of expression (2) will reveal other interesting aspects of exponential smoothing. First we will expand this equation:

$$F_{t+1} = \alpha X_t + (1-\alpha)[\alpha X_{t-1} + (1-\alpha)F_{t-1}]$$

$$= \alpha X_t + \alpha(1-\alpha)X_{t-1} + (1-\alpha)^2 F_{t-1}$$

Additional expansion will yield

$$F_{t+1} = X_t + \alpha(1-\alpha)X_{t-1} + \alpha(1-\alpha)X_{t-2}^2 + \ldots \tag{3}$$

Looking at the coefficients of the time series, we can see that they decrease exponentially. For instance, if $\alpha = .5$, then equation (3) becomes

$$F_{t+1} = .5X_t + .25X_{t-1} + .125X_{t-2} + \ldots \tag{4}$$

Thus, the name exponential smoothing is applied. In control theory this expression is sometimes called a first order filter.

The coefficient in expression (4) can be interpreted as weights applied to each observation. The more recent observations receive the most weight. This is often desirable in time-series forecasting because the most recent observations may be more indicative of the future. Consequently, another advantage of exponential smoothing is that the more recent data receive the most weight.

If equation (2) is rearranged, we can see another interpretation:

$$F_{t+1} = F_t + \alpha(X_t - F_t)$$

The last term in parentheses is the error of the last forecast. Therefore,

$$F_{t+1} = F_t + \propto E_t \tag{5}$$

This says that the new forecast is obtained by adjusting the old forecast by some portion of the last forecast error.

We can see that if $\propto$ is close to 1, the forecast error has a substantial effect on the new forecast. However, if $\propto$ is close to 0, the error has little impact. Therefore, if the underlying process is level but has substantial random variation, a small $\propto$ will "filter" out most of this random variable. In fact, most computerized inventory control systems seldom use a smoothing constant greater than .3.

10.2 Exponential Smoothing with a Trend

If the underlying process is horizontal, single exponential smoothing is appropriate. However, if a trend is involved, forecasts using equation (5) will lag behind the actual observations. Consequently, an adjustment must be made for the trend.

There is more than one way to include trend considerations in an exponential smoothing model. Holt's (see Makridakis (1978)) method will be used here:

$$F_t = \propto X_t + (1-\propto)(F_{t-1}+b_{t-1}) \tag{6}$$

$$b_t = \beta(F_t - F_{t-1}) + (1-\beta)\, b_{t-1} \tag{7}$$

$$F_{t+h} = F_t + b_t h \tag{8}$$

where b_t = estimate of the trend for period t,

β = trend smoothing constant, $0 \leq \beta \leq 1$, and

F_{t+h} = forecast for period $t+h$ (a forecast lead time of h).

Equation (6) smooths the old forecast, F_{t-1}, adjusted by the trend, b_{t-1}, to obtain a forecast, F_t, for period t. The estimate of the trend is smoothed in expression (7). This is done by assuming that the part of the difference in the last two forecasts ($F_t - F_{t-1}$) is due to changes in the trend. The last expression, (8), provides a forecast h periods ahead by adding to the forecast for period t the trend per period multiplied by the number of periods ahead to be forecast. The number of periods to be forecast ahead is called the *forecast lead time*. The number of periods forecast beyond the last observation in a time series is called the *forecast horizon*.

10.3 Seasonal Exponential Smoothing

A time series often exhibits seasonal patterns. Examples are the sales of toys, snow tires, and ice cream. The difference between a *seasonal* pattern and a *cyclical* pattern is that a seasonal pattern repeats itself in short intervals such as a week, month, or year. A cyclical pattern has a much larger duration. One approach to analyzing time-series data that contains seasonal patterns is to remove the seasonal component. Until this is done, it is difficult to distinguish between random effects and the seasonal effects. Several exponential smoothing methods have been developed which consider seasonal patterns; one of these is Winter's method (which is explained by Montgomery (1976)).

Before presenting the equations for this method, we need to develop a procedure for calculating seasonal indices. These indices will be used to remove the seasonal component in a time series. Consider the following seasonal sales pattern of units sold: 100, 200, and 300, which is repeated every 3 periods. A seasonal index can be computed as follows:

(1) Compute an average sales/period:
$(100 + 200 + 300)/3 = 200$

(2) Compute seasonal index, I_t, for each period by dividing the sales per period by the average sales per period.
$I_1 = 100/200 + .5$
$I_2 = 200/200 = 1.0$
$I_3 = 300/200 = 1.5$

The sum of the indices should equal the number of periods in a season. Using these indices we can remove the seasonal effect from the data. For instance, removing the seasonal component from the sales for period 1 results in:

$$X_1' = X_1/I_1,$$

$$= 100/.5 = 200 \text{ units}$$

This procedure will be used to remove the seasonal effects from a time series.

Now we can look at the equations for Winter's method which considers trend and seasonality:

$$F_t = \alpha X_t/I_{t-k} + (1-\alpha)(F_{t-1}+b_{t-1}) \tag{9}$$

$$b_t = \beta(F_t - F_{t-1}) + (1-\beta)b_{t-1} \tag{10}$$

$$I_t = \gamma X_t/F_t + (1-\gamma)I_{t-k} \tag{11}$$

$$F_{t+h} = (F_t + b_t h)I_{t-k+h} \tag{12}$$

where K = the number of periods in a season

I_t = the seasonal index for period t, and

γ = the seasonal index smoothing constant, $0 \leq \gamma \leq 1$.

Equation (9) is the same as equation (6) used in Holt's method except that X_t is divided by the seasonal index to remove the seasonal fluctuations. The result of equation (9), F_t, is a smoothed estimate for period t. This is obtained by adding the trend to the forecast for the period $t-1$ and by removing the seasonability fluctuations.

Equation (10) is the same as Holt's equation (7). The next equation, (11), smooths the seasonal indices. As the seasonal indices are smoothed, they may no longer sum to the number of periods in a season. In the program that is presented later in the chapter, seasonal indices are normalized at the end of each season so that the sum of the indices equals the number of periods in a season. Equation (12) provides a forecast for period $t+1$. This forecast includes adjustments for trend and seasonality.

To implement Winter's method, it is desirable to have some procedures for estimating initial values for each of the components of the model (permanent, trend, and seasonal). Montgomery (1976) provides procedures to estimate initial values for each component.

These procedures will be presented here. It is assumed that the number of data elements used for initialization represents at least two seasons. The initial trend component may be estimaed by:

$$b_0 = (\bar{X}_m - \bar{X}_1)/(m - 1)K$$

where m = number of seasons

$\bar{X}_m$ = average of the observations in season m (13)

K = length of a season

b_0 = initial estimate for the trend

This expression evaluates the differences in the average value between the first season and the last season, $(\bar{X}_m - \bar{X}_1)$. This value is then divided by the number of seasons between these averages to obtain the trend per season. It is assumed that $\bar{X}_i$ occurs in the middle of a season.

The initial permanent component may be estimated by

$$a_0 = \bar{X}_1 - (K/2)b_0 \tag{14}$$

Again, it is assumed that $\bar{X}_1$ occurs at the middle of the first season. Therefore, by subtracting $K/2b_0$ (½ of a season times the trend), we can obtain an estimate for the initial permanent component at the beginning of period 1.

Seasonal indices must be computed for each time period in the initialization time series. The procedure used is similar to the seasonal index calculations presented above. The following expression is used.

$$I_t = X_t/(\bar{X}_i - ((K+1)/2 - j)b_0,\ j = 1, 2, \ldots, K;\ t = 1, 2, \ldots, mK, \tag{15}$$

where j is the position of data element X_i within a season. Thus, if there are three periods in a season, X_3 would be in position 3 in the first season and X_4 would be in position 1 in the second season. The average seasonal value is adjusted by removing the estimated trend, leaving an estimate for the permanent component. Consequently, equation (15) is the ratio of actual observations to the adjusted average seasonal observation for that season.

Equation (15) will result in K indices for each season. Using these estimates, an average seasonal index is computed:

$$\bar{I}_t = 1/m \sum_{i=0}^{m-1} I_{t+i+k},\ t = 1, 2, \ldots, K \tag{16}$$

The sum of the average indices should equal m. To insure this, the indices are normalizd.

$$I_t(0) = \bar{I}_t \left(m \Big/ \sum_{t-1}^{m} \bar{I}_t \right),\quad t = 1, 2, \ldots, K \tag{17}$$

Having determined initial values for the model components (permanent, trend, and seasonal), we can then apply Winter's method to forecast values in a time series.

10.4 Exponential Smoothing Program

Figure 10.1 contains an exponential smoothing program incorporating Winter's method. The name of the program is EXSMOOTH. Some examples will be presented to illustrate how to use this program. A time series having as many as 100 elements and 24 periods in a season can be studied. A larger problem can be studied by changing the program dimension statements.

You can enter data in two ways, from the keyboard or from a previously saved disk file. The following data management options are provided by the program:

1. List time series data.
2. Change time series data.
3. Delete data from time series.
4. Add to time series.
5. Store time series data on disk.

After you have entered the data, the exponential smoothing computations will be performed when the PERFORM FORECAST COMPUTATIONS option is specified. These computations are organized into two major parts: initialization phase and forecast phase.

You may use the initialization phase to develop estimates for the model components (permanent, trend, and seasonal) and to optimize the values for the smoothing constants associated with each of the component types. Note that at least two seasons must be available for the program to develop these estimates. Also, the number of periods used in initialization must be a multiple of the season length.

These initial estimates are used in the initialization phase. During this phase, the values are smoothed so the best values will be used to start the forecasting phase. At the end of this phase, the program displays a comparison of forecast versus actual data for each period used in the initialization phase. The user may input initial values for the model components and the smoothing constants. In this case the initialization phase is used only to smooth these input values to obtain the best values to be used in the forecast phase.

During the forecast phase, forecasts are made for all periods in the time series beyond the last period used in the initialization phase. If a forecast horizon is specified (periods beyond the last period in the time series), forecasts are also made for these periods. The results of the initialization and forecast phases can be displayed on your CRT or your printer. If they are displayed on your CRT, program execution is periodically interrupted to give you an opportunity to read the results before they are scrolled off the screen. Details of using this program will now be illustrated with some examples. In these examples, program output is denoted by **bold** characters and user input is denoted by underlined characters.

EXAMPLE 1. A local firm manufactures turbo chargers for trucks. This firm would like some help in forecasting impeller wheel failures for the next two quarters. They have six years (24 quarters) of historical data representing failures by quarter.

```
1000  REM  TIME SERIES FORECASTING PROGRAM--WINTER'S METHOD
1010  REM  THIS PROGRAM WILL PERFORM TIME SERIES FORECASTING USING
1020  REM  THE WINTER'S METHOD OF EXPONENTIAL SMOOTHING. A TIME
1030  REM  SERIES HAVING AS MANY AS 100 OBSERVATIONS AND 24 PERIODS
1040  REM  IN A SERIES CAN BE STUDIED. LARGER PROBLEMS CAN BE
1050  REM  STUDIED BY CHANGING THE PROGRAM DIMENSION STATEMENTS.
1060  REM  ADDITIONAL INSTRUCTIONS ARE PROVIDED DURING PROGRAM
1070  REM  EXECUTION. DEFINATIONS OF MAJOR PROGRAM VARIABLES:
1080  REM        LM=MAXIMUM NUMBER PERIODS IN TIME SERIES
1090  REM        LP=MAXIMUM NUMBER PERIODS IN A SEASON
1100  REM        LN=MAXIMUM NUMBER OF SEASONS IN INITIALIZATION
1110  REM        IT=NUMBER OF INITIALIZATION PERIODS
1120  REM        LT=FORECAST LEAD TIME
1130  REM        IZN=FORECAST HORIZON
1140  REM        X(I)=TIME SERIES DATA
1150  REM        A(I)=PERMANENT COMPONENT
1160  REM        B(I)=TREND COMPONENT
1170  REM        SF(I)=SEASON FACTOR
1180  REM        FCS(I)=FORECAST
1190  REM        ER(I)=ERROR IN FORECAST
1200 DOS$ =  CHR$ (4): REM  CTRL-D
1210 LM = 100: REM  MAX NO. PERIODS IN TIME SERIES
1220 LP = 24: REM  MAX NO. PERIODS IN SEASON
1230 LN = 24: REM  MAX NO. SEASONS IN INITIALIZATION
1240  DIM X(LM),A(LM),B(LM),SF(LP),AXS(LN),RSF(LN,LP)
1250  DIM ER(LM),FCS(LM),SS(LM),TEMP(LP)
1260  PRINT : PRINT
1270  PRINT "****************************************"
1280  PRINT "*"; TAB( 39);"*"
1290  PRINT "*"; TAB( 5);"TIME SERIES FORECASTING PROGRAM   *"
1300  PRINT "*"; TAB( 13);"WINTER'S METHOD           *"
1310  PRINT "*"; TAB( 39);"*"
1320  PRINT "****************************************"
1330  PRINT : PRINT
1340  INPUT "WANT PROGRAM INSTRUCTIONS (Y/N) ? ";Y$
1350  IF Y$ <  > "Y" THEN 1610
1360  PRINT : PRINT  TAB( 10);"PROGRAM INSTRUCTIONS"
1370  PRINT  TAB( 9);"==========================": PRINT
1380  PRINT "   TIME SERIES DATA MAY BE ENTERED FROM": PRINT
1390  PRINT "KEYBOARD OR DISK. TIME SERIES DATA": PRINT
1400  PRINT "MANAGEMENT OPTIONS PROVIDED :"
1410  PRINT : PRINT  TAB( 5);"- LIST DATA"
1420  PRINT  TAB( 5);"- CHANGE DATA"
1430  PRINT  TAB( 5);"- ADD TO DATA"
1440  PRINT  TAB( 5);"- DELETE DATA"
1450  PRINT  TAB( 5);"- STORE DATA ON DISK": PRINT
1460  PRINT "USER MUST SPECIFY NUMBER OF DATA": PRINT
1470  PRINT "ELEMENTS TO BE USED FOR MODEL": PRINT
1480  PRINT "INITIALIZATION, NUMBER OF PERIODS IN": PRINT
1490  PRINT "SEASON, FORECAST LEAD TIME, AND FORECAST HORIZON": PRINT
1500  INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
1510  PRINT "HORIZON. USER HAS THE OPTION OF": PRINT
1520  PRINT "INPUTTING MODEL COMPONENTS (PERMANENT,": PRINT
1530  PRINT "TREND, AND SEASONAL FACTORS) AND": PRINT
1540  PRINT "SMOOTHING CONSTANTS (ONE CONSTANT FOR": PRINT
1550  PRINT "EACH TYPE OF MODEL COMPONENT) OR ": PRINT
1560  PRINT "SPECIFYING THAT THE PROGRAM DETERMINE": PRINT
1570  PRINT "THESE VALUES. PROGRAM WILL OPTIMIZE": PRINT
1580  PRINT "SMOOTHING CONSTANTS BY PERFORMING A": PRINT
1590  PRINT "GRID SEARCH USING SPECIFIED LIMITS AND": PRINT
1600  PRINT "INCREMENTS WITHIN THESE LIMITS.": PRINT
1610  PRINT : INPUT "ENTER DATA FROM KEYBOARD OR DISK (K/D)?";K$
1620  IF (K$ <  > "K") AND (K$ <  > "D") THEN  PRINT "INVALID": GOTO 1610
1630  IF K$ = "K" THEN  GOSUB 7170
1640  IF K$ = "D" THEN  GOSUB 5960
1650  PRINT  TAB( 3);"PROGRAM OPTIONS:": PRINT
```

```
1660  PRINT  TAB( 6);"1- LIST TIME SERIES DATA"
1670  PRINT  TAB( 6);"2- CHANGE TIME SERIES DATA"
1680  PRINT  TAB( 6);"3- DELETE DATA FROM TIME SERIES"
1690  PRINT  TAB( 6);"4- ADD TO TIME SERIES"
1700  PRINT  TAB( 6);"5- STORE TIME SERIES DATA ON DISK"
1710  PRINT  TAB( 6);"6- PERFORM FORECAST COMPUTATIONS"
1720  PRINT  TAB( 6);"7- STUDY NEW MODEL"
1730  PRINT  TAB( 6);"8- QUIT"
1740  PRINT : INPUT "OPTION ? ";IOP: PRINT
1750  IF (IOP < 1) OR (IOP > 8) THEN  PRINT "INVALID": GOTO 1650
1760  IF IOP = 1 THEN  GOSUB 6570
1770  IF IOP = 2 THEN  GOSUB 6930
1780  IF IOP = 3 THEN  GOSUB 7500
1790  IF IOP = 4 THEN  GOSUB 7020
1800  IF IOP = 5 THEN  GOSUB 5860
1810  IF IOP = 6 THEN 1850
1820  IF IOP = 7 THEN  CLEAR : GOTO 1210
1830  IF IOP = 8 THEN  PRINT  TAB( 10);"*** END OF PROGRAM ***": END
1840  HOME : GOTO 1650
1850  PRINT "NUMBER OF DATA ELEMENTS USED FOR"
1860  INPUT "INITIALIZATION ? ";IT
1870  IF (IT < 0) AND (IT >  = LD) THEN  PRINT "INVALID": GOTO 1850
1880  PRINT : INPUT "NUMBER PERIODS IN A SEASON ? ";LS
1890  IF (LS < 0) OR (LS > IT) THEN  PRINT "INVALID": GOTO 1880
1900  IF LS = 0 THEN LS = 1
1910 XL = LS
1920  IF (LS <  = LP) THEN  GOTO 1940
1930  PRINT : PRINT "WILL NOT ACCEPT A SEASON OF THIS LENGTH": GOTO 1880
1940  PRINT : PRINT "ESTIMATION OF MODEL PARAMETERS DESIRED"
1950  INPUT "(Y/N) ? ";PY$
1960  IF (PY$ <  > "Y") AND (PY$ <  > "N") THEN  PRINT "INVALID": GOTO 1940
1970  IF PY$ = "N" THEN 2040: REM  ASK FOR INITIAL COMPONENTS
1980 S =  INT (IT / LS)
1990 IS = (IT - (S * LS))
2000  IF (PY$ = "Y") AND (IS = 0) THEN 2130
2010  PRINT "ESTIMATION OF MODEL PARAMETERS REQUIRES THAT NUMBER"
2020  PRINT "OF INITIALIZATION PERIODS BE A MULTIPLE OF SEASON LENGTH"
2030  GOTO 1850
2040  PRINT : PRINT "ENTER INITIAL PERMANENT AND TREND"
2050  INPUT "COMPONENTS ? ";AO,BO: PRINT
2060  PRINT "ENTER INITIAL SEASONAL FACTORS :"
2070  PRINT "--------------------------------"
2080  FOR I = 1 TO LS
2090  PRINT "SEASONAL FACTOR ";I;" ";
2100  INPUT SF(I)
2110 TEMP(I) = SF(I)
2120  NEXT I
2130  PRINT : INPUT "SMOOTHING CONSTANT OPTIMIZATION DESIRED (Y/N) ? ";SY$
2140  IF (SY$ <  > "Y") AND (SY$ <  > "N") THEN  PRINT "INVALID": GOTO 2130
2150  IF SY$ = "Y" THEN 2260
2160  PRINT : PRINT "ENTER PERMANENT COMPONENT SMOOTHING"
2170  INPUT "FACTOR-ALPHA ? ";AP
2180  IF (AP < 0) OR (AP > 1) THEN  PRINT "INVALID": GOTO 2160
2190  PRINT : PRINT "ENTER TREND COMPONENT SMOOTHING"
2200  INPUT "FACTOR-BETA ? ";BE
2210  IF (BE < 0) OR (BE > 1) THEN  PRINT "INVALID"; GOTO 2190
2220  PRINT : PRINT "ENTER SEASONAL COMPONENT SMOOTHING"
2230  INPUT "FACTOR-GAMMA ? ";GA
2240  IF (GA < 0) OR (GA > 1) THEN  PRINT "INVALID"; GOTO 2220
2250  GOTO 2550
2260  PRINT : PRINT "ENTER LOWER LIMIT,UPPER LIMIT,STEP SIZE"
2270  PRINT "FOR ALPHA--PERMANENT COMPONENT"
2280  PRINT "SMOOTHING FACTOR ";
2290  INPUT AL,AU,AI
2300  PRINT
2310  IF (AL < 0) OR (AU < 0) OR (AI < 0) THEN  PRINT "INVALID": GOTO 2260
```

```
2320  IF (AL > 1) OR (AU > 1) OR (AI > 1) THEN  PRINT "INVALID": GOTO 2260
2330  PRINT "ENTER LOWER LIMIT,UPPER LIMIT,STEP SIZE"
2340  PRINT "FOR BETA--TREND COMPONENT SMOOTHING"
2350  PRINT "FACTOR ";
2360  INPUT BL,BU,BI
2370  PRINT
2380  IF (BL < 0) OR (BU < 0) OR (BI < 0) THEN  PRINT "INVALID": GOTO 2330
2390  IF (BL > 1) OR (BU > 1) OR (BI > 1) THEN  PRINT "INVALID": GOTO 2330
2400  PRINT "ENTER LOWER LIMIT,UPPER LIMIT,STEP SIZE"
2410  PRINT "FOR GAMMA--SEASONAL COMPONENT SMOOTHING"
2420  PRINT "FACTOR ";
2430  INPUT GL,GU,GI
2440  PRINT
2450  IF (GL < 0) OR (GU < 0) OR (GI < 0) THEN  PRINT "INVALID": GOTO 2400
2460  IF (GL > 1) OR (GU > 1) OR (GI > 1) THEN  PRINT "INVALID": GOTO 2400
2470 P1 = (AU - AL) / AI + 1
2480 P2 = (BU - BL) / BI + 1
2490 P3 = (GU - GL) / GI + 1
2500 T = P1 * P2 * P3
2510  PRINT T;" TRIALS ARE REQUIRED TO FIND OPTIMUM"
2520  PRINT "VALUES FOR SMOOTHING CONSTANTS."
2530  INPUT "OKAY TO CONTINUE (Y/N) ? ";Y$
2540  IF Y$ = "N" THEN 2260
2550  PRINT : INPUT "ENTER FORECAST LEAD TIME ? ";LT
2560  IF LT < 0 THEN  PRINT "INVALID"; GOTO 2550
2570  PRINT : INPUT "NUMBER PERIODS IN FORECAST HORIZON ? ";IZN
2580  IF IZN < 0 THEN  PRINT "INVALID": GOTO 2570
2590 XTL = LT:ID = IT
2600  IF IT > (LD - LT) THEN ID = LD - LT
2610  PRINT : PRINT "WOULD YOU LIKE TO SEE INPUT DATA"
2620  INPUT "DISPLAYED (Y/N) ? ";Y$
2630  IF Y$ <  > "Y" THEN 2650
2640  GOSUB 6070
2650  PRINT : INPUT "ARE DATA CORRECT (Y/N) ? ";Y$
2660  IF Y$ = "Y" THEN 2700
2670  PRINT : INPUT "DO YOU WANT TO REENTER DATA (Y/N) ? ";Y$: PRINT
2680  IF Y$ = "Y" THEN 1850
2690  HOME : GOTO 1650
2700  PRINT : PRINT "RESULTS TO APPEAR AT CRT OR PRINTER"
2710  INPUT "(C/P) ? ";C$
2720  IF (C$ <  > "C") AND (C$ <  > "P") THEN  PRINT "INVALID": GOTO 2700
2730  IF C$ = "C" THEN 2820
2740 X$ = "************************************************"
2750  PRINT DOS$;"PR#1"
2760  PRINT  TAB( 5);X$
2770  PRINT  TAB( 5);"*"; TAB( 48);"*"
2780  PRINT  TAB( 5);"*"; TAB( 5);"TIME SERIES FORECASTING-WINTER'S METHOD"; TAB( 5);"*"
2790  PRINT  TAB( 5);"*"; TAB( 48);"*"
2800  PRINT  TAB( 5);X$
2810  PRINT : PRINT
2820  HOME : PRINT DOS$;"PR#0"
2830  IF PY$ = "N" THEN 3420
2840  PRINT : PRINT "INITIAL VALUES FOR PERMANENT, TREND,"
2850  PRINT "AND SEASONAL COMPONENTS TO BE ESTIMATED"
2860  PRINT "FROM DATA"
2870 IK = IT / LS: REM  NO. OF SEASONS
2880 I1 = 1
2890 I2 = LS
2900  FOR I = 1 TO IK
2910 AXS(I) = 0
2920  FOR J = I1 TO I2
2930 AXS(I) = AXS(I) + X(J)
2940  NEXT J
2950 AXS(I) = AXS(I) / XL: REM  AVERAGE X PER SEASON I
2960 I1 = I2 + 1
2970 I2 = I1 + LS - 1
```

```
2980  NEXT I
2990 R = IT - LS
3000 BO = (AXS(IK) - AXS(1)) / R: REM  ESTIMATE TREND COMPONENT
3010 AO = AXS(1) - (XL - 1) / 2 * BO: REM  ESTIMATE PERMANENT COMPONENT
3020 IO = 0
3030  FOR I1 = 1 TO IK
3040  FOR I2 = 1 TO LS
3050 IJ = I2 + IO * LS
3060 R = I2
3070  REM  CALCULATE SEASONAL VALUES FOR EACH SEASON
3080 RSF(I1,I2) = X(IJ) / (AXS(I1) - (((XL + 1) / 2) - R) * BO)
3090  NEXT I2
3100 IO = IO + 1
3110  NEXT I1
3120 GTT = 0
3130 R = IK
3140  FOR J = 1 TO LS
3150 TT = 0
3160  FOR I = 1 TO IK
3170 TT = TT + RSF(I,J)
3180  NEXT I
3190  REM  AVERAGE SEASONAL FACTOR FOR PERIOD J IN A SEASON.
3200 SF(J) = TT / R
3210  REM  TOTAL OF ALL SEASONAL FACTORS
3220 GTT = GTT + SF(J)
3230  NEXT J
3240  IF C$ = "C" THEN 3260
3250  PRINT DOS$;"PR#1"
3260 AO =  INT (AO * 10 ^ 4 + .5) /  INT (10 ^ 4 + .5)
3270  PRINT : PRINT "ESTIMATED PERMANENT COMPONENT = ";AO
3280  PRINT : PRINT "ESTIMATED TREND COMPONENT = ";BO
3290  PRINT : PRINT "PERIOD       ESTIMATED SEASONAL FACTOR": PRINT
3300  FOR J = 1 TO LS
3310  REM  NORMALIZE SEASONAL FACTORS
3320 SF(J) = SF(J) * (XL / GTT)
3330 TEMP(J) = SF(J)
3340  IF C$ = "P" THEN 3390
3350 IC = 1
3360  IF J <  > (IC * 15) THEN 3390
3370 IC = IC + 1
3380  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
3390  PRINT J,SF(J)
3400  NEXT J
3410  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
3420  IF SY$ = "N" THEN 4050
3430  PRINT DOS$;"PR#0"
3440  PRINT : PRINT : PRINT  TAB( 4);"SMOOTHING CONSTANT OPTIMIZATION"
3450  PRINT : PRINT : PRINT  TAB( 11);"PROGRAM EXECUTING!"
3460  PRINT  TAB( 10);"==================": PRINT
3470 IA = P1:IB = P2:IG = P3
3480 MER = 1E + 30
3490 AP = AL:BE = BL:GA = GL
3500 A1 = AL
3510  FOR II = 1 TO IA
3520 AM = 1 - A1
3530 B1 = BL
3540  FOR IJ = 1 TO IB
3550 BM = 1 - B1
3560 G1 = GL
3570  FOR IK = 1 TO IG
3580  FOR IL = 1 TO LS
3590 SF(IL) = TEMP(IL)
3600  NEXT IL
3610 GM = 1 - G1
3620 A(1) = A1 * (X(1) / SF(1)) + AM * AO
3630 B(1) = B1 * (A(1) - (AO - BO)) + BM * BO
```

```
3640 SF(1) = G1 * (X(1) / A(1)) + GM * SF(1)
3650 IH = 1 + LT
3660  IF IH <  = LS THEN 3680
3670 IH = IH - LS: GOTO 3660
3680 EST = (A(1) + XTL * B(1)) * SF(IH)
3690 DI = (X(1 + LT) - EST) ^ 2
3700  FOR I = 2 TO ID
3710 S =  INT (I / LS)
3720 IP = (I - (S * LS))
3730  IF IP = 0 THEN IP = LS
3740 A(I) = A1 * (X(I) / SF(IP)) + AM * (A(I - 1) + B(I - 1))
3750 B(I) = B1 * (A(I) - A(I - 1)) + BM * B(I - 1)
3760 SF(IP) = G1 * (X(I) / A(I)) + GM * SF(IP)
3770  IF IP <  > LS THEN 3850
3780 GTT = 0
3790  FOR J = 1 TO LS
3800 GTT = GTT + SF(J)
3810  NEXT J
3820  FOR J = 1 TO LS
3830 SF(J) = SF(J) * (XL / GTT)
3840  NEXT J
3850 IH = IP + LT
3860  IF IH <  = LS THEN 3880
3870 IH = IH - LS: GOTO 3860
3880 EST = (A(I) + XTL * B(I)) * SF(IH)
3890 DI = DI + (X(I + LT) - EST) ^ 2
3900  NEXT I
3910  IF DI > MER THEN 3940
3920 AP = A1:BE = B1:GA = G1
3930 MER = DI
3940 G1 = G1 + GI
3950  NEXT IK
3960 B1 = B1 + BI
3970  NEXT IJ
3980 A1 = A1 + AI
3990  NEXT II
4000  IF C$ = "P" THEN  PRINT DOS$;"PR#1"
4010  PRINT : PRINT "OPTIMUM SMOOTHING CONSTANTS :": PRINT
4020  PRINT "ALPHA = ";AP; TAB( 15);"BETA = ";BE; TAB( 28);"GAMMA = ";GA
4030  PRINT : PRINT "SUM OF ERRORS SQUARED = ";MER
4040  IF C$ = "C" THEN  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
4050  IF LT = 0 THEN LT = 1: REM  START MODEL INITIALIZATION
4060  FOR I = 1 TO LT
4070 ER(I) = 0:FCS(I) = 0
4080  NEXT I
4090  REM  PERFORM SMOOTHING AND CALCULATE FORECASTS
4100  FOR I = 1 TO LS
4110 SF(I) = TEMP(I)
4120  NEXT I
4130 AM = 1 - AP
4140 BM = 1 - BE
4150 GM = 1 - GA
4160 A(1) = AP * (X(1) / SF(1)) + AM * A0
4170 B(1) = BE * (A(1) - (A0 - B0)) + BM * B0
4180 SF(1) = GA * (X(1) / A(1)) + GM * SF(1)
4190 IH = 1 + LT
4200  IF IH <  = LS THEN 4220
4210 IH = IH - LS: GOTO 4200
4220 FCS(1 + LT) = (A(1) + XTL * B(1)) * SF(IH)
4230 SS(1) = SF(1)
4240 ER(1 + LT) = X(1 + LT) - FCS(1 + LT)
4250 SUMSQ = ER(1) ^ 2
4260 XMAD =  ABS (ER(1))
4270  FOR I = 2 TO IT
4280 S =  INT (I / LS)
4290 IL = (I - (S * LS))
```

```
4300  IF IL = 0 THEN IL = LS
4310 A(I) = AP * (X(I) / SF(IL)) + AM * (A(I - 1) + B(I - 1))
4320 B(I) = BE * (A(I) - A(I - 1)) + BM * B(I - 1)
4330 SF(IL) = GA * (X(I) / A(I)) + GM * SF(IL)
4340 SS(I) = SF(IL)
4350  IF IL <  > LS THEN 4440
4360  REM  NORMALIZE SEASONAL FACTORS
4370 GTT = 0
4380  FOR J = 1 TO LS
4390 GTT = GTT + SF(J)
4400  NEXT J
4410  FOR J = 1 TO LS
4420 SF(J) = SF(J) * (XL / GTT)
4430  NEXT J
4440 IH = IL + LT
4450  IF IH <  = LS THEN 4470
4460 IH = IH - LS: GOTO 4450
4470 SUMSQ = SUMSQ + ER(I) ^ 2
4480 XMAD = XMAD +  ABS (ER(I))
4490  IF (I + LT) > LD THEN 4520
4500 FCS(I + LT) = (A(I) + XTL * B(I)) * SF(IH)
4510 ER(I + LT) = X(I + LT) - FCS(I + LT)
4520  NEXT I
4530 XMAD = XMAD / (IT - LT):MSE = SUMSQ / (IT - LT)
4540  IF C$ = "P" THEN  PRINT DOS$;"PR#1"
4550  PRINT : PRINT  TAB( 5);"RESULTS OF INITIALIZATION PHASE"
4560  PRINT  TAB( 4);"================================="
4570  PRINT : PRINT "SEASON LENGTH=";LS,"FORECAST LEAD TIME=";LT
4580  PRINT : PRINT  TAB( 10);"ACTUAL"
4590  PRINT "PERIOD    DATA    FCS.VALUE    ERROR"
4600  PRINT "-----------------------------------------"
4610 IC = 1
4620  FOR I = 1 TO IT
4630 FCS(I) =  INT (FCS(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
4640 ER(I) =  INT (ER(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
4650  IF C$ = "P" THEN  GOTO 4690
4660  IF I <  > (IC * 15) THEN 4690
4670 IC = IC + 1
4680  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
4690  HTAB 3: PRINT I;: HTAB 11: PRINT X(I);: HTAB 19: PRINT FCS(I);: HTAB 31: PRINT ER(
4700  NEXT I
4710  IF C$ = "P" THEN  GOTO 4730
4720  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
4730  PRINT : PRINT  TAB( 12);"MODEL COMPONENTS"
4740  PRINT  TAB( 11);"================="
4750  PRINT : PRINT "PERIOD   PERMANENT    TREND    SEASONAL"
4760  PRINT "-----------------------------------------"
4770 IC = 1
4780  FOR I = 1 TO IT
4790 A(I) =  INT (A(I) * 10 ^ 4 + .5) /  INT (10 ^ 4 + .5)
4800 B(I) =  INT (B(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
4810 SS(I) =  INT (SS(I) * 10 ^ 7 + .5) /  INT (10 ^ 7 + .5)
4820  IF C$ = "P" THEN  GOTO 4860
4830  IF I <  > (IC * 16) THEN 4860
4840  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
4850 IC = IC + 1
4860  HTAB 3: PRINT I;: HTAB 10: PRINT A(I);: HTAB 21: PRINT B(I);: HTAB 31: PRINT SS(I)
4870  NEXT I
4880  PRINT : PRINT "MEAN SQUARED ERROR = ";MSE
4890  PRINT : PRINT "MEAN ABSOLUTE DEVIATION = ";XMAD
4900  PRINT DOS$;"PR#0"
4910 I1 = IT + 1: REM  START FORECAST PHASE
4920  IF (IT = LD) AND (IZN = 0) THEN 5830
4930  IF IT = LD THEN 5200
4940 SUMSQ = 0
4950 XMAD = 0
```

```
4960  REM  PERFORM SMOOTHING AND CALCULATE FORECASTS
4970  FOR I = I1 TO LD
4980 S =  INT (I / LS)
4990 IL = (I - (S * LS))
5000  IF IL = 0 THEN IL = LS
5010 FCS(I) = (A(I - LT) + XTL * B(I - LT)) * SF(IL)
5020 A(I) = AP * (X(I) / SF(IL)) + AM * (A(I - 1) + B(I - 1))
5030 B(I) = BE * (A(I) - A(I - 1)) + BM * B(I - 1)
5040 SF(IL) = GA * (X(I) / A(I)) + GM * SF(IL)
5050 SS(I) = SF(IL)
5060  IF IL <  > LS THEN 5150
5070  REM  NORMALIZE SEASONAL FACTORS
5080 GTT = 0
5090  FOR J = 1 TO LS
5100 GTT = GTT + SF(J)
5110  NEXT J
5120  FOR J = 1 TO LS
5130 SF(J) = SF(J) * (XL / GTT)
5140  NEXT J
5150 ER(I) = X(I) - FCS(I)
5160 SUMSQ = SUMSQ + ER(I) ^ 2
5170 XMAD = XMAD +  ABS (ER(I))
5180  NEXT I
5190 XMAD = XMAD / (LD - IT):MSE = SUMSQ / (LD - IT)
5200  IF C$ = "P" THEN  PRINT DOS$;"PR#1": GOTO 5220
5210  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
5220  PRINT : PRINT  TAB( 6);"RESULTS OF FORECASTING PHASE"
5230  PRINT  TAB( 5);"=============================="
5240  PRINT : PRINT  TAB( 10);"ACTUAL"
5250  PRINT "PERIOD    DATA    FCS.VALUE    ERROR"
5260  PRINT "-----------------------------------"
5270  IF IT = LD THEN 5380
5280 IC = 1
5290  FOR I = I1 TO LD
5300 FCS(I) =  INT (FCS(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
5310 ER(I) =  INT (ER(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
5320  IF C$ = "P" THEN  GOTO 5360
5330  IF (I - I1 + 1) <  > (IC * 20) THEN 5360
5340 IC = IC + 1
5350  PRINT : INPUT "PRESS RETURN TO CONTINUE";Y$: PRINT
5360  HTAB 3: PRINT I;: HTAB 11: PRINT X(I);: HTAB 19: PRINT FCS(I);: HTAB 31: PRINT ER(I)
5370  NEXT I
5380  IF IZN = 0 THEN  GOTO 5580
5390  FOR I = LD + 1 TO LD + IZN
5400 S =  INT (I / LS)
5410 IL = (I - (S * LS))
5420  IF IL = 0 THEN IL = LS
5430  IF (I - LT) > LD THEN 5460
5440 FCS(I) = (A(I - LT) + XTL * B(I - LT)) * SF(IL)
5450  GOTO 5470
5460 FCS(I) = (A(LD) + (I - LD) * B(LD)) * SF(IL)
5470  NEXT I
5480  PRINT : PRINT "FORECAST HORIZON": PRINT
5490 IE1 = LD - I1 + 1
5500  FOR I = LD + 1 TO LD + IZN
5510  IF C$ = "P" THEN  GOTO 5560
5520 IE1 = IE1 + 1
5530  IF IE1 <  > (IC * 20) THEN 5560
5540 IC = IC + 1
5550  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
5560  HTAB 3: PRINT I;: HTAB 19: PRINT FCS(I)
5570  NEXT I
5580  IF IT = LD THEN 5820
5590  IF C$ = "P" THEN  GOTO 5610
5600  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
5610  PRINT : PRINT  TAB( 12);"MODEL COMPONENTS"
```

```
5620  PRINT  TAB( 11);"==========": PRINT
5630  PRINT "PERIOD    PERMANENT     TREND     SEASONAL"
5640  PRINT "----------------------------------------"
5650 IC = 1
5660  FOR I = I1 TO LD
5670 A(I) =  INT (A(I) * 10 ^ 4 + .5) /  INT (10 ^ 4 + .5)
5680 B(I) =  INT (B(I) * 10 ^ 5 + .5) /  INT (10 ^ 5 + .5)
5690 SS(I) =  INT (SS(I) * 10 ^ 7 + .5) /  INT (10 ^ 7 + .5)
5700  IF C$ = "P" THEN  GOTO 5740
5710  IF (I - I1 + 1) <  > (IC * 20) THEN 5740
5720 IC = IC + 1
5730  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
5740  HTAB 3: PRINT I;: HTAB 10: PRINT A(I);: HTAB 21: PRINT B(I);: HTAB 31: PRINT SS(I)
5750  NEXT I
5760  IF C$ = "P" THEN  GOTO 5780
5770  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
5780  PRINT : PRINT "FORECAST PHASE :"
5790  PRINT "----------------"
5800  PRINT : PRINT "MEAN SQUARED ERROR = ";MSE
5810  PRINT : PRINT "MEAN ABSOLUTE DEVIATION = ";XMAD
5820  PRINT DOS$;"PR#0"
5830  PRINT : PRINT : PRINT  TAB( 12);"<<< END OF RUN >>>"
5840  PRINT : PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
5850  GOTO 1650
5860  REM  SUBROUTINE: STORE DATA ON DISK
5870  PRINT : INPUT "ENTER NAME OF DISK:FILE ? ";NAM$
5880  PRINT DOS$;"OPEN";NAM$
5890  PRINT DOS$;"WRITE";NAM$
5900  PRINT LD
5910  FOR I = 1 TO LD
5920  PRINT X(I)
5930  NEXT I
5940  PRINT DOS$;"CLOSE";NAM$
5950  PRINT : RETURN
5960  REM  SUBROUTINE: READ DATA FROM DISK
5970  PRINT : INPUT "ENTER NAME OF DISK:FILE ? ";NAM$
5980  PRINT DOS$;"OPEN";NAM$
5990  PRINT DOS$;"READ";NAM$
6000  INPUT LD
6010  FOR I = 1 TO LD
6020  INPUT X(I)
6030  NEXT I
6040  PRINT DOS$;"CLOSE";NAM$
6050  PRINT : PRINT "TIME SERIES CONTAINS ";LD;" DATA ELEMENTS": PRINT
6060  RETURN
6070  REM  SUBROUTINE: LIST MODEL DATA
6080  HOME : PRINT  TAB( 10);"LISTING OF INPUT DATA"
6090  PRINT  TAB( 9);"----------": PRINT
6100  PRINT "# OF DATA POINTS IN TIME SERIES = ";LD
6110  PRINT : PRINT "NUMBER OF DATA POINTS USED FOR"
6120  PRINT "INITIALIZATION = ";IT
6130  PRINT : PRINT "NUMBER OF PERIODS IN A SEASON = ";LS
6140  PRINT : PRINT "FORECAST LEAD TIME = ";LT
6150  PRINT : PRINT "PERIODS IN FORECAST HORIZON = ";IZN
6160  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
6170  IF SY$ = "Y" THEN 6220
6180  PRINT : PRINT  TAB( 11);"SMOOTHING CONSTANTS"
6190  PRINT  TAB( 10);"----------------------"
6200  PRINT : PRINT "ALPHA = ";AP;"   ";"BETA = ";BE;"   ";"GAMMA = ";GA: PRINT
6210  GOTO 6290
6220  PRINT : PRINT  TAB( 8);"SMOOTHING CONSTANT RANGES"
6230  PRINT  TAB( 7);"--------------------------"
6240  PRINT  TAB( 10);"LOWER     UPPER"
6250  PRINT "TYPE     LIMIT     LIMIT      INCREMENT"
6260  PRINT : PRINT "ALPHA"; TAB( 12);AL; TAB( 22);AU; TAB( 34);AI
```

```
6270  PRINT : PRINT "BETA"; TAB( 12);BL; TAB( 22);BU; TAB( 34);BI
6280  PRINT : PRINT "GAMMA"; TAB( 12);GL; TAB( 22);GU; TAB( 34);GI: PRINT
6290  IF PY$ = "Y" THEN  RETURN
6300  PRINT "PERMANENT AND TREND COMPONENTS: ";A0;" , ";B0
6310  IF LS = 0 THEN  RETURN
6320  PRINT : PRINT "PERIODS"; TAB( 16);"** SEASONAL FACTOR **"
6330 S =  INT (LS / 4)
6340 IR = (LS - (S * 4))
6350 IP =  INT (LS / 4)
6360  IF (IR <  > 0) THEN IP = IP + 1
6370  FOR I = 1 TO IP
6380 I1 = I:I2 = I + 3
6390  IF (I <  > 11) THEN 6420
6400  INPUT "PRESS RETURN TO CONTINUE";Y$
6410  PRINT : PRINT "PERIODS"; TAB( 16);"** SEASONAL FACTOR **"
6420  INPUT "TIME SERIES ? ";EG,EN: PRINT
6430  IF (I = IP) AND (IR = 0) THEN 6540
6440  ON IR GOTO 6450,6480,6510
6450 I2 = I1
6460  PRINT : PRINT I1;" THRU ";I2;" :"; TAB( 15);SF(I)
6470  GOTO 6550
6480 I2 = I1 + 1
6490  PRINT : PRINT I1;" THRU ";I2;" :"; TAB( 15);SF(I); TAB( 21);SF(I + 1)
6500  GOTO 6550
6510 I2 = I1 + 2
6520  PRINT : PRINT I1;" THRU ";I2;" :"; TAB( 15);SF(I); TAB( 21);SF(I + 1); TAB(
     27);SF(I + 2)
6530  GOTO 6550
6540  PRINT : PRINT I1;" THRU ";I2;" :"; TAB( 15);SF(I); TAB( 21);SF(I + 1); TAB(
     27);SF(I + 2); TAB( 33);SF(I + 3)
6550  NEXT I
6560  RETURN
6570  REM  SUBROUTINE: LIST TIME SERIES DATA
6580  HOME
6590  PRINT : PRINT "  LISTING OF TIME SERIES DATA ELEMENTS"
6600  PRINT "---------------------------------------"
6610  PRINT "ELEMENT NOS."; TAB( 22);"** DATA VALUES **": PRINT
6620 IC = 1
6630 S =  INT (LD / 4)
6640 IR = (LD - (S * 4))
6650 IP =  INT (LD / 4)
6660  IF (IR <  > 0) THEN IP = IP + 1
6670  FOR I = 1 TO IP
6680 I1 = 4 * (I - 1) + 1:I2 = I1 + 3
6690  IF I <  > (IC * 10) THEN 6730
6700 IC = IC + 1
6710  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$
6720  PRINT : PRINT "ELEMENT NOS."; TAB( 22);"** DATA VALUES **": PRINT
6730  IF I < IP THEN 6880
6740  IF (I = IP) AND (IR = 0) THEN 6880
6750  ON IR GOTO 6760,6800,6840
6760 I2 = I1
6770  PRINT "ELEMENT ";I1;" :"
6780  PRINT  TAB( 21);X(I1)
6790  GOTO 6900
6800 I2 = I1 + 1
6810  PRINT "ELEMENTS ";I1;" THRU ";I2;" : "
6820  PRINT  TAB( 21);X(I1); TAB( 26);X(I1 + 1)
6830  GOTO 6900
6840 I2 = I1 + 2
6850  PRINT "ELEMENTS ";I1;" THRU ";I2;" : "
6860  PRINT  TAB( 21);X(I1); TAB( 26);X(I1 + 1); TAB( 31);X(I1 + 2)
6870  GOTO 6900
6880  PRINT "ELEMENTS ";I1;" THRU ";I2;" : "
6890  PRINT  TAB( 21);X(I1); TAB( 26);X(I1 + 1); TAB( 31);X(I1 + 2); TAB( 36);
     X(I1 + 3)
6900  NEXT I
```

```
6910  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
6920  RETURN
6930  REM  SUBROUTINE: CHANGE DATA
6940  PRINT
6950  PRINT : PRINT "ENTER POSITION OF DATA ELEMENT TO BE"
6960  INPUT "CHANGED ? ";I
6970  IF (I < 0) OR (I > LD) THEN  PRINT "INVALID": GOTO 6950
6980  PRINT : INPUT "ENTER DATA ELEMENT ? ";X(I)
6990  PRINT : INPUT "CHANGE OTHER DATA ELEMENTS (Y/N) ? ";Y$
7000  IF Y$ = "Y" THEN 6950
7010  RETURN
7020  REM  SUBROUTINE: ADD TO TIME SERIES
7030  PRINT
7040  PRINT "TIME SERIES NOW CONTAINS ";LD;" DATA"
7050  PRINT "ELEMENTS": PRINT
7060  INPUT "HOW MANY DATA ELEMENTS ARE TO BE ADDED ? ";ADD
7070  IF (ADD < 0) THEN  PRINT "MUST BE >=0"; GOTO 7060
7080  IF ADD = 0 THEN  RETURN
7090  IF ((ADD + LD) > LM) THEN  PRINT "TOTAL MUST BE <=";LM: GOTO 7060
7100  PRINT : PRINT "ENTER DATA"
7110  FOR I = LD + 1 TO LD + ADD
7120  PRINT : PRINT "VALUE FOR DATA ELEMENT ";I;" = ";
7130  INPUT X(I)
7140  NEXT I
7150 LD = LD + ADD
7160  PRINT : RETURN
7170  REM  SUBROUTINE: ENTER TIME SERIES DATA FROM KEYBOARD
7180  INPUT "NUMBER OF DATA POINTS IN TIME SERIES ?";LD: PRINT
7190  IF (LD < 0) OR (LD > LM) THEN  PRINT "INVALID": GOTO 7180
7200  PRINT "ENTER TIME SERIES DATA ELEMENTS"
7210  PRINT "(4 ELEMENTS PER LINE)": PRINT
7220 S =  INT (LD / 4)
7230 IR = (LD - (S * 4))
7240 IP =  INT (LD / 4)
7250  IF (IR <  > 0) THEN IP = IP + 1
7260  FOR I = 1 TO IP
7270 I1 = 4 * (I - 1) + 1:I2 = I1 + 3
7280  IF I < IP THEN 7430
7290  IF (I = IP) AND (IR = 0) THEN 7430
7300  ON IR GOTO 7310,7350,7390
7310 I2 = I1
7320  PRINT "ELEMENT ";I2;
7330  INPUT X(I1)
7340  GOTO 7450
7350 I2 = I1 + 1
7360  PRINT "ELEMENTS ";I1;" THRU ";I2;
7370  INPUT X(I1),X(I1 + 1)
7380  GOTO 7450
7390 I2 = I1 + 2
7400  PRINT "ELEMENTS ";I1;" THRU ";I2;
7410  INPUT X(I1),X(I1 + 1),X(I1 + 2)
7420  GOTO 7450
7430  PRINT "ELEMENTS ";I1;" THRU ";I2;
7440  INPUT X(I1),X(I1 + 1),X(I1 + 2),X(I1 + 3)
7450  NEXT I
7460  PRINT : INPUT "LIST DATA ELEMENTS (Y/N) ? ";Y$
7470  IF Y$ = "N" THEN  RETURN
7480  GOSUB 6570
7490  RETURN
7500  REM  SUBROUTINE: DELETE DATA ELEMENTS
7510  PRINT
7520  PRINT "ENTER BEGINNING AND ENDING POSITION OF"
7530  PRINT "DATA ELEMENTS TO BE DELETED FROM"
7540  INPUT "TIME SERIES ? ";EG,EN
7550  IF (EG < 0) OR (EN < 0) THEN  PRINT "INVALID": GOTO 7520
7560  IF (EG > LD) OR (EN > LD) THEN  PRINT "INVALID": GOTO 7520
```

```
7570  IF (EN - EG + 1) < 0 THEN  PRINT "INVALID": GOTO 7520
7580  IF EN = LD THEN 7620
7590  FOR I = EN + 1 TO LD
7600 X(I - (EN - EG + 1)) = X(I)
7610  NEXT I
7620 LD = LD - (EN - EG + 1)
7630  PRINT "DATA ELEMENTS HAVE BEEN DELETED"
7640  PRINT : RETURN
```

Figure 10.1—Exponential Smoothing Forecasting Program

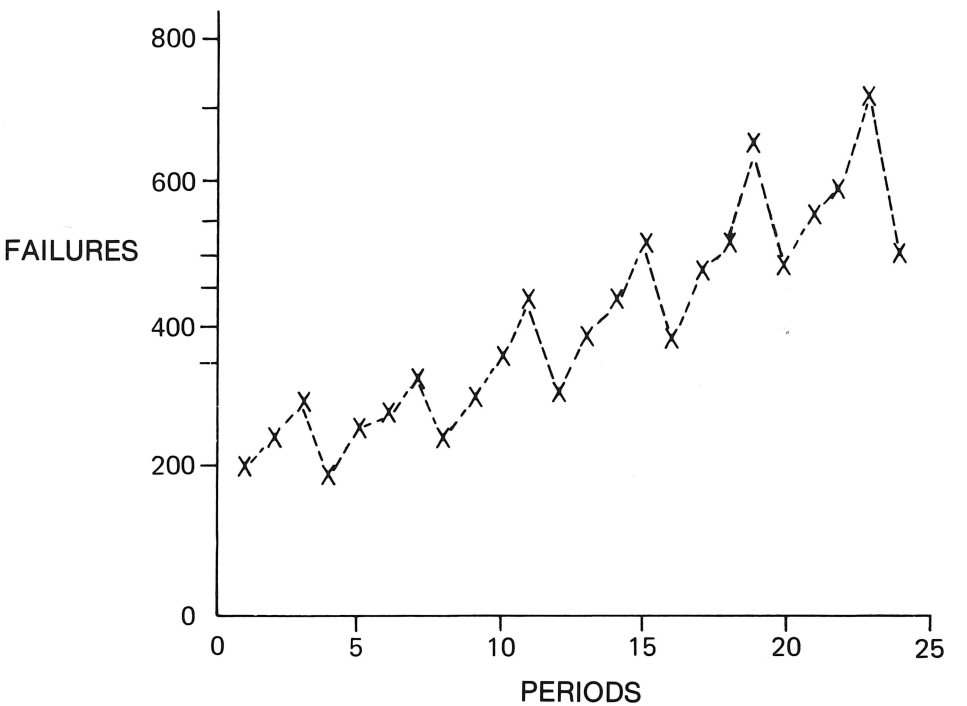


Figure 10.2—Impeller Wheel Failures Example

Year 1	202	242	293	195
Year 2	257	281	343	244
Year 3	307	359	438	318
Year 4	397	437	528	379
Year 5	481	529	626	445
Year 6	564	601	725	507

Figure 10.2 is a plot of this data. Note that trend and seasonal effects are exhibited.

SOLUTION. The Time Series Forecasting Program is stored in the file EXSMOOTH.

```
LOAD EXSMOOTH
RUN

**********************************
*                                *
* TIME SERIES FORECASTING PROGRAM *
*         WINTER'S METHOD        *
*                                *
**********************************

WANT PROGRAM INSTRUCTIONS (Y/) ? Y

      PROGRAM INSTRUCTIONS
     ====================

TIME SERIES DATA MAY BE ENTERED FROM

KEYBOARD OR DISK. TIME SERIES DATA

MANAGEMENT OPTIONS PROVIDED:

      -LIST DATA
      -CHANGE DATA
      -ADD TO DATA
      -DELETE DATA
      -STORE DATA ON DISK

USER MUST SPECIFY NUMBER OF DATA

ELEMENTS TO BE USED FOR MODEL

INITIALIZATION, NUMBER OF PERIODS IN

SEASON, FORECAST LEAD TIME, AND FORECAST HORIZON

PRESS RETURN TO CONTINUE ->

USER HAS THE OPTION OF

INPUTTING MODEL COMPONENTS (PERMANENT,

TREND, AND SEASONAL FACTORS) AND

SMOOTHING CONSTANTS (ONE CONSTANT FOR

EACH TYPE OF MODEL COMPONENT) OR

SPECIFYING THAT THE PROGRAM DETERMINE

THESE VALUES. PROGRAM WILL OPTIMIZE

SMOOTHING CONSTANTS BY PERFORMING A

GRID SEARCH USING SPECIFIED LIMITS AND
```

```
INCREMENTS WITHIN THESE LIMITS.

ENTER DATA FROM KEYBOARD OR DISK (K/D)?K
NUMBER OF DATA POINTS IN TIME SERIES ?24

ENTER TIME SERIES DATA ELEMENTS
(4 ELEMENTS PER LINE)

ELEMENTS 1 THRU 4? 202,242,293,195
ELEMENTS 5 THRU 8? 257,281,343,244
ELEMENTS 9 THRU 12? 307,359,438,318
ELEMENTS 13 THRU 16? 397,437,528,379
ELEMENTS 17 THRU 20? 481,529,626,445
ELEMENTS 21 THRU 24? 564,601,725,507

LIST DATA ELEMENTS (Y/N) ? Y

 LISTING OF TIME SERIES DATA ELEMENTS

     ELEMENT NOS.        **DATA VALUES**
ELEMENTS 1 THRU 4:
                      202    242    293    195
ELEMENTS 5 THRU 8:
                      257    281    343    244
ELEMENTS 9 THRU 12:
                      307    359    438    318
ELEMENTS 13 THRU 16:
                      397    437    528    379
ELEMENTS 17 THRU 20:
                      481    529    626    445
ELEMENTS 21 THRU 24:
                      564    601    725    507

PRESS RETURN TO CONTINUE ->

      PROGRAM OPTIONS
            1-LIST TIME SERIES DATA
            2-CHANGE TIME SERIES DATA
            3-DELETE DATA FROM TIME SERIES
            4-ADD TO TIME SERIES
            5-STORE TIME SERIES DATA ON DISK
            6-PERFORM FORECAST COMPUTATIONS
            7-STUDY NEW MODEL
            8-QUIT
```

Having entered the data and verified that they are correct, we will store them in a disk file named WINEXAMP. Then the forecast computations will be performed using the first 16 periods to initialize the model. The remaining 8 periods can be used to evaluate the model. Since little is known about the time series being studied, we will let the program estimate the model parameters and determine the best smoothing constants.

```
OPTION ? 5

ENTER NAME OF DISK:FILE ? WINEXAMP
```

```
    PROGRAM OPTIONS
       1-LIST TIME SERIES DATA
       2-CHANGE TIME SERIES DATA
       3-DELETE DATA FROM TIME SERIES
       4-ADD TO TIME SERIES
       5-STORE TIME SERIES DATA ON DISK
       6-PERFORM FORECAST COMPUTATIONS
       7-STUDY NEW MODEL
       8-QUIT

OPTION ? 6

NUMBER OF DATA ELEMENTS USED FOR
INITIALIZATION ?  16

NUMBER PERIODS IN A SEASON ? 4

ESTIMATION OF MODEL PARAMETERS DESIRED
(Y/N) ? Y

SMOOTHING CONSTANT OPTIMIZAITON DESIRED
(Y/N) ? Y
```

To determine the optimum smoothing constants, the program performs a grid search using the lower and upper limits and step size specified for each constant. Remember that the smoothing constants must have a value in the range of 0 to 1.

In this example, lower and upper limits of .1 and .3 and a step size of .1 were specified for each of the smoothing constants. Consequently, 27 different combinations are possible (.1, .2, .3 for each of the constants, giving $3 \times 3 \times 3 = 27$). Note that you should be careful and not make the number of trials too large. For instance, five minutes of computer time may be required to evaluate 127 combinations (trials). Each combination of smoothing constants is evaluated by using each one as initial values in the initialization phase. The combination resulting in the smallest value for the sum of errors squared is chosen as the optimum.

```
ENTER LOWER LIMIT,UPPER LIMIT,STEP SIZE
FOR ALPHA--PERMANENT COMPONENT
SMOOTHING FACTOR ?.1,.3,.1

ENTER LOWER LIMIT,UPPER LIMIT,STEP SIZE
FOR BETA--TREND COMPONENT SMOOTHING
FACTOR ?.1,.3,.1

ENTER LOWER LIMIT,UPPER LIMIT,STEP SIZE
FOR GAMMA--SEASONAL COMPONENT SMOOTHING
FACTOR ?.1,.3,.1

27 TRIALS ARE REQUIRED TO FIND OPTIMUM
VALUES FOR SMOOTHING CONSTANTS.
OKAY TO CONTINUE (Y/N) ? Y
```

We will assume a forecast lead time of 1 period (1 quarter) and a forecast horizon of 2 periods (each period represents 1 quarter).

```
ENTER FORECAST LEAD TIME ? 1
```

```
NUMBER PERIODS IN FORECAST HORIZON ? 2

WOULD YOU LIKE TO SEE INPUT DATA
DISPLAYED (Y/N) ? Y

   LISTING OF INPUT DATA

# OF DATA POINTS IN TIME SERIES = 24

NUMBER OF DATA POINTS USED FOR
INITIALIZATION = 16

NUMBER OF PERIODS IN A SEASON = 4

FORECAST LEAD TIME = 1

PERIODS IN FORECAST HORIZON = 2

PRESS RETURN TO CONTINUE ->
```

	SMOOTHING CONSTANT RANGES		
TYPE	LOWER LIMIT	UPPER LIMIT	INCREMENT
ALPHA	.1	.3	.1
BETA	.1	.3	.1
GAMMA	.1	.3	.1

```
ARE DATA CORRECT (Y/N) ? Y

RESULTS TO APPEAR AT CRT OR PRINTER
(C/P) ? C

INITIAL VALUES FOR PERMANENT, TREND,
AND SEASONAL COMPONENTS TO BE ESTIMATED
FROM DATA

ESTIMATED PERMANENT COMPONENT 207.7188
ESTIMATED TREND COMPONENT = 16.8541667

 PERIOD          ESTIMATED SEASONAL FACTOR
   1                    .966179903
   2                   1.03880288
   3                   1.1947823
   4                    .800234919

PRESS RETURN TO CONTINUE ->

   SMOOTHING CONSTANT OPTIMIZATION

      PROGRAM EXECUTING!

OPTIMUM SMOOTHING CONSTANTS :

ALPHA= .3      BETA= .3      GAMMA= .1

SUM OF ERRORS SQUARED= 2298.49985

PRESS RETURN TO CONTINUE ->
```

Note that the smoothing constants for the permanent and trend components are at their specified upper limits and the seasonal smoothing constant is at the lower limit. Look at the values for the model components displayed below; you will note that the permanent and trend components are not very stable. The model is trying to adjust to this instability by using larger values for the associated smoothing constants. Remember that a small smoothing constant is used for a stable process.

RESULTS OF INITIALIZATION PHASE

SEASON LENGTH=4 FORECAST LEAD TIME=1

PERIOD	ACTUAL DATA	FCS VALUE	ERROR
1	202	0	0
2	242	233.83478	8.16522
3	293	292.89082	.10918
4	195	210.35014	-15.35014
5	257	264.0346	-7.0346
6	281	298.11233	-17.11233
7	343	352.6972	-9.6972
8	244	243.48223	.51777
9	307	308.62267	-1.62267
10	359	344.73603	14.26397
11	438	418.3526	19.6474
12	318	295.79835	22.20165
13	397	382.9903	14.00967
14	437	437.84668	-.84668

PRESS RETURN TO CONTINUE ->

15	528	526.64773	1.35227
16	379	368.20607	10.79393

PRESS RETURN TO CONTINUE ->

MODEL COMPONENTS

PERIOD	PERMANENT	TREND	SEASONAL
1	208.1244	16.97585	.9666193
2	227.4583	17.68327	1.041317
3	245.169	17.69149	1.1948135
4	257.1059	15.96511	.7960557
5	270.8884	15.31032	.9650905
6	281.2702	13.83176	1.0373684
7	292.6678	13.10154	1.1928514
8	305.9644	13.16006	.7964122
9	318.6211	13.00905	.9667353
10	335.7467	14.244	1.0424934
11	354.9217	15.72332	1.1992
12	378.9909	18.22706	.8021644
13	401.577	19.53477	.966624
14	420.8674	19.46148	1.0395991
15	440.6681	19.56324	1.1962474

```
PRESS RETURN TO CONTINUE ->
   16          464.2788          20.77749          .801673

MEAN SQUARED ERROR = 142.618865

MEAN ABSOLUTE DEVIATION = 9.514953

PRESS RETURN TO CONTINUE ->

   RESULTS OF FORECASTING PHASE

                ACTUAL
PERIOD          DATA          FCS. VALUE          ERROR
-------------------------------------------------------
   17            481          468.38185          12.61815
   18            529          530.61277          -1.61277
   19            626          636.07722          -10.07722
   20            445          441.10696           3.89304
   21            564          554.9629            9.037122
   22            601          621.71248          -20.71248
   23            725          732.11509          -7.11509
   24            507          506.07328            .92672

FORECAST HORIZON
   25                         631.790207
   26                         696.543004
PRESS RETURN TO CONTINUE ->
```

Looking at the forecast phase results, we see that 632 and 697 impeller wheel failures are predicted to occur during the next two quarters (25th and 26th periods). Since we do not know of any changes in the process being modeled, these numbers will be used as our forecast of failures for these periods.

```
        MODEL COMPONENTS

PERIOD        PERMANENT          TREND          SEASONAL
   17          488.9765         21.95355          .9674301
   18          510.4642         21.81379         1.0383022
   19          529.7481         21.05484         1.1936779
   20          552.2613         21.49235          .8013369
   21          576.5566         22.33322          .9683466
   22          592.9042         20.53753         1.0356629
   23          611.6532         20.00097         1.1926408
   24          632.0012         20.10508          .8012899

PRESS RETURN TO CONTINUE ->

FORECAST PHASE :

MEAN SQUARE ERROR = 105.085526

MEAN ABSOLUTE DEVIATION = 8.24907272

              <<<END OF RUN >>>
```

```
      PROGRAM OPTIONS

PRESS RETURN TO CONTINUE ->

  PROGRAM OPTIONS

      1-LIST TIME SERIES DATA
      2-CHANGE TIME SERIES DATA
      3-DELETE DATA FROM TIME SERIES
      4-ADD TO TIME SERIES
      5-STORE DATA ON DISK
      6-PERFORM FORECAST COMPUTATIONS
      7-STUDY NEW MODEL
      8-QUIT

OPTION ? 8

      *** END OF PROGRAM ***
```

EXAMPLE 2. Two quarters have passed and the company that manufactures turbo chargers was very pleased with our projections of impeller wheel failures for this time period. As a result, this firm would like our assistance in developing a forecast for the next year (4 quarters). In reviewing the failure data, errors were found in the figures for periods 23 and 24. Also, the failure figures for periods 25 and 26 are now available:

PERIOD	FAILURE
23	730 (revised data)
24	510 (revised data)
25	640
26	700

SOLUTIONS. Since the original time series containing 24 quarters of failure data was saved in a disk file named WINEXAMP, we can enter this data using the enter data from disk option. However, the data associated with periods 23 and 24 must be changed and data for periods 25 and 26 must be added to the time series.

```
RUN

************************************
*                                  *
*  TIME SERIES FORECASTING PROGRAM *
*        WINTER'S METHOD           *
*                                  *
************************************

WANT PROGRAM INSTRUCTIONS (Y/N) ? N

ENTER DATA FROM KEYBOARD OR DISK (K/D)?D

ENTER NAME OF DISK:FILE? WINEXAMP

TIME SERIES CONTAINS 24 DATA ELEMENTS
```

```
      PROGRAM OPTIONS

            1-LIST TIME SERIES DATA
            2-CHANGE TIME SERIES DATA
            3-DELETE DATA FROM TIME SERIES
            4-ADD TO TIME SERIES
            5-STORE TIME SERIES DATA ON DISK
            6-PERFORM FORECAST COMPUTATIONS
            7-STUDY NEW MODEL
            8-QUIT
```

There are two ways we could revise the data for periods 23 and 24. One way is to use option 2, CHANGE TIME SERIES DATA to change the incorrect data elements. Another way is to use option 3, DELETE DATA FROM TIME SERIES, in conjunction with option 4, ADD TO TIME SERIES. The latter approach was used here so that option 3 could be illustrated.

```
OPTION ? 3

ENTER BEGINNING AND ENDING POSITIONS OF
DATA ELEMENTS TO BE DELETED FROM
TIME SERIES? 23,24
DATA ELEMENTS HAVE BEEN DELETED

      PROGRAM OPTIONS:

            1-LIST TIME SERIES DATA
            2-CHANGE TIME SERIES DATA
            3-DELETE DATA FROM TIME SERIES
            4-ADD TO TIME SERIES
            5-STORE DATA ON DISK
            6-PERFORM FORECAST COMPUTATIONS
            7-STUDY NEW MODEL
            8-QUIT

OPTION ? 4


TIME SERIES NOW CONTAINS 22 DATA
ELEMENTS

HOW MANY DATA ELEMENTS ARE TO BE ADDED? 4

ENTER DATA

VALUE FOR DATA ELEMENT 23 = ?730

VALUE FOR DATA ELEMENT 24 = ?510

VALUE FOR DATA ELEMENT 25 = ?640

VALUE FOR DATA ELEMENT 26 = ?700
```

```
      PROGRAM OPTIONS:

            1-LIST TIME SERIES DATA
            2-CHANGE TIME SERIES DATA
            3-DELETE DATA FROM TIME SERIES
            4-ADD TO TIME SERIES
            5-STORE TIME SERIES DATA ON DISK
            6-PERFORM FORECAST COMPUTATIONS
            7-STUDY NEW MODEL
            8-QUIT

OPTION ? 1

  LISTING OF TIME SERIES DATA ELEMENTS

    ELEMENT NOS.      **DATA ELEMENT VALUES**
ELEMENTS  1 THRU  4:
                      202    242    293    195
ELEMENTS  5 THRU  8:
                      257    281    343    244
ELEMENTS  9 THRU 12:
                      307    359    438    318
ELEMENTS 13 THRU 16:
                      397    437    528    379
ELEMENTS 17 THRU 20:
                      481    529    626    445
ELEMENTS 21 THRU 24:
                      564    601    730    510
ELEMENTS 24 THRU 26:
                      640    700

PRESS RETURN TO CONTINUE ->
```

Looking at the listing of the time series data, we see that the data are correct. Therefore, we are ready to perform the forecast computations. Our model will be initialized using the first 20 time periods. Remember that the number of time periods used for initialization must be a multiple of the number of periods in a season (the season length is 4). Also, initial estimates for the model components can be obtained from example 1; these estimates will be used here. Likewise, the optimum values from example 1 will be input for the smoothing constants.

```
      PROGRAM OPTIONS:

          1-LIST TIME SERIES DATA
          2-CHANGE TIME SERIES DATA
          3-DELETE DATA FROM TIME SERIES
          4-ADD TO TIME SERIES
          5-STORE TIME SERIES DATA ON DISK
          6-PERFORM FORECAST COMPUTATIONS
          7-STUDY NEW MODEL
          8-QUIT

OPTION ? 6

NUMBER OF DATA ELEMENTS USED
FOR INITIALIZATION ? 20
```

```
NUMBER PERIODS IN A SEASON ? 4
ESTIMATION OF MODEL PARAMETERS DESIRED
(Y/N) ? N

ENTER INITIAL PERMANENT AND TREND
COMPONENTS? 208,17

ENTER INITIAL SEASONAL FACTORS:

SEASONAL FACTOR 1 ? .97

SEASONAL FACTOR 2 ? 1.04

SEASONAL FACTOR 3 ? 1.19

SEASONAL FACTOR 4 ? .80

SMOOTHING CONSTANT OPTIMIZATION DESIRED
(Y/N) ? N

ENTER PERMANENT COMPONENT SMOOTHING
FACTOR-ALPHA ? .3

ENTER TREND COMPONENT SMOOTHING
FACTOR-BETA ? .3

ENTER SEASONAL COMPONENT SMOOTHING
FACTOR-GAMMA ? .1

ENTER FORECAST LEAD TIME? 1

NUMBER PERIODS IN FORECAST HORIZON ? 4

WOULD YOU LIKE TO SEE INPUT DATA
DISPLAYED (Y/N) ? N

ARE DATA CORRECT (Y/N) ? Y

RESULTS TO APPEAR AT CRT OR PRINTER
(C/P) ? P
```

We would like our results displayed at the printer.

```
*****************************
*                           *
*  TIME SERIES FORECASTING  *
*      WINTER'S METHOD      *
*                           *
*****************************


 RESULTS OF INITIALIZATION PHASE
==============================

SEASON LENGTH=4 FORECAST LEAD TIME=1
```

PERIOD	ACTUAL DATA	FCS VALUE	ERROR
1	202	0	0
2	242	234.10035	7.89965
3	293	291.64654	-1.35346
4	195	210.58447	-15.58447
5	257	265.34912	-8.34912
6	281	298.38242	-17.38242
7	343	351.18698	-8.18698001
8	244	243.60028	.39972
9	307	309.88328	-2.88328
10	359	344.88951	14.11049
11	438	416.54661	21.45339
12	318	295.84721	22.15279
13	397	384.35781	12.64219
14	437	437.99494	-.99494
15	528	524.53452	3.46548
16	379	368.21316	10.78684
17	481	469.89633	11.10367
18	529	530.8119	-1.81192
19	626	633.75233	-7.75233
20	445	441.09464	3.90536

PERIOD	MODEL COMPONENTS PERMANENT	TREND	SEASONAL
1	208.0742	17.02227	.9700807
2	227.3752	17.70589	1.042432
3	245.4223	17.80825	1.190386
4	257.3864	16.005	.7957616
5	270.8603	15.28066	.968248
6	281.1402	13.78043	1.0384533
7	292.858	13.16165	1.188828
8	306.1703	13.20685	.7961196
9	318.4857	12.9394	.9696401
10	335.493	14.15977	1.04357
11	355.0552	15.7805	1.1955448
12	379.166	18.27961	.8018749
13	401.3674	19.45615	.969275
14	420.5368	19.37012	1.0406385
15	440.7788	19.63169	1.1929265
16	464.4569	20.84559	.8013754
17	488.7428	21.87768	.9698449
18	510.0976	21.72081	1.0392944
19	529.8667	21.13532	1.1906465
20	552.4656	21.57439	.8010266

MEAN SQUARED ERROR = 126.088095

MEAN ABSOLUTE DEVIAITON = 9.06413183

RESULTS OF FORECASTING PHASE
==========

```
                ACTUAL
PERIOD          DATA          FCS VALUE          ERROR
-------------------------------------------------------
  21             564          556.61672          7.38328
  22             601          621.97866         -20.97866
  23             730          729.68209           .31791
  24             510          507.36053          2.63947
  25             640          636.1669           3.8331
  26             700          702.59438         -2.59438

FORECAST HORIZON

  27                          830.848416
  28                          575.896954
  29                          718.246498
  30                          788.272173

     MODEL COMPONENTS
     ================

PERIOD        PERMANENT          TREND          SEASONAL
-------------------------------------------------------
  21          576.3243          22.25969          .9705448
  22          592.5271          20.44262         1.036605
  23          613.0498          20.46666         1.190441
  24          634.5053          20.76328          .8011551
  25          656.453           21.11862          .9712570
  26          676.821           20.89344         1.0366618

FORECAST PHASE:

MEAN SQUARED ERROR = 87.1846914
MEAN ABSOLUTE DEVIATION = 6.29113355


      <<<END OF RUN>>>

PRESS RETURN TO CONTINUE ->
```

Looking at the results, we see that the following failures are predicted:

Period	Failures
27	831
28	576
29	718
30	788

Again, since we know of no changes in the process being modeled and because our previous predictions were very good, we will use the above figures for our failures forecast.

```
      PROGRAM OPTIONS

        1-LIST TIME SERIES DATA
        2-CHANGE TIME SERIES DATA
        3-DELETE DATA FROM TIME SERIES
        4-ADD TO TIME SERIES
```

```
        5-STORE TIME SERIES DATA ON DISK
        6-PERFORM FORECAST COMPUTATIONS
        7-STUDY NEW MODEL
        8-QUIT

OPTION ? 8

        ***END OF PROGRAM***
```

10.5 Summary

In this chapter, exponential smoothing, a popular time series forecasting technique, was presented. Winter's method for incorporating trend and seasonal considerations into a model was implemented in a computer program. Several data management options were also incorporated into the program to enhance its usefulness. When using time series forecasting techniques you should remember that historical observations are being projected into the future. Therefore, these techniques are normally used for short term forecasting. A plot of the historical data can also be a valuable aid in developing a forecast. The plot subroutine from Chapter 2 can make this task easier.

References

Makridakis, Spyros, and Steven C. Wheelwright, *Forecasting Methods and Applications*, John Wiley & Sons, New York, 1978.

Montgomery, Douglas C., and Lynwood A. Johnson, *Forecasting and Time Series Analysis*, McGraw-Hill, New York, 1976.

Exercises

1. The following data depicts the monthly average stock price of a microcomputer firm.

Month	Average Stock Price
1	$3.5
2	3.6
3	3.2
4	3.7
5	4.2
6	4.8
7	5.1
8	5.4
9	5.8

   a. Use a 3-period moving average to estimate the stock price for the 10th month.
   b. Use single exponential smoothing with $\alpha = .3$ to estimate the stock price for the 10th month.

2. The unit sales of 1200 baud modems for the Speciality Modems Corporation have been increasing. The sales data for the last 18 months are given below. You are to develop a unit sales forecast for the next 6 months (months 19–24) using Winter's exponential smoothing method. Also, you are to plot this data using the plot subroutine from Chapter 2.

Month	Sales (1000's of Units)
1	11
2	9
3	11

4	13
5	15
6	16
7	14
8	17
9	17
10	18
11	20
12	22
13	21
14	24
15	27
16	29
17	30
18	33

3. For the time series of 1, 3, 5, 7, 9, 11, 13, 15, 17, compute a forecast for period 10 using:
   a. a 3-period moving average
   b. single exponential smoothing with $\propto = .5$.
   c. Holt's method of exponential smoothing with a trend.
      Let $\propto$ and $\beta = .1$
   d. Winter's method (use program provided in this chapter).
   Which is the better method? Why?

4. For the time series of 1, 4, 7, 10, 1, 4, 7, 10, 1, 4, 7, 10, 1, 4, 7, 10, compute a forecast for period 17 using:
   a. Single exponential smoothing with $\propto = .5$
   b. Winter's method (use program provided in this chapter).

5. The time series representing the number of units of microcomputers selling for less than $500 has a seasonal pattern. The unit sales for the last 24 months are given below. You are to develop a sales forecast for the next 6 months using Winter's method. Also, you are to plot this data.

Month	Sales (1000's of Units)
1	18
2	19
3	22
4	19
5	20
6	24
7	22
8	24
9	26
10	28
11	32
12	34
13	24
14	24
15	26
16	26
17	29
18	32
19	32
20	36
21	40
22	44
23	46
24	50

11

Project Planning & Scheduling with CPM

The critical path method (CPM) is a popular planning and scheduling technique. Working together, Remington Rand and DuPont developed CPM around 1960. At about the same time, a similar method, called program evaluation and review technique (PERT), was developed by the Navy and Lockheed Aircraft Corporation. CPM is the most popular and very similar to PERT; consequently, CPM will be used in this chapter. The interested reader can refer to Moder (1964) and Weist (1977) for an in-depth discussion of both techniques.

11.1 The CPM Technique

One objective of CPM is to develop a schedule such that a project will be completed on time. This technique requires a list of all activities that make up a project, an estimated duration time for each activity, and a list of any activities that must be completed before a particular activity can be started (immediate predecessors). Table 11.1 contains an example of the required information. Note that activities B and C cannot start unitl activity A has been completed. Therefore, activity A is an immediate predecessor to activities B and C. We can also say that B and C are immediate successors to activity A.

The data from Table 11.1 can be used to draw a CPM network such as the one depicted in Figure 11.1. The circles in the network are called nodes (or events) and each is numbered. The arrow between two nodes represents an activity. Thus, we can say activity A begins at node 1 and ends at node 2. The letter on each arrow in Figure 11.1 denotes a particular activity and the number denotes the activity duration. In addition to representing a particular activity, the arrows show which activities must precede other activities. In Figure 11.1 activity A must precede activities B and C.

A dashed line in a CPM network represents a ''dummy'' activity. Such an activity has no duration time; it is only used to denote precedence relationships. In Figure 11.1, the activity between nodes 7 and 8 is a dummy activity showing that activity I cannot start until activity H has been completed. Activity H cannot be represented by an arrow between

Table 11.1
ACQUIRE A COMPUTER
PROJECT ACTIVITIES

Activity	Description	Duration (weeks)	Immediate Predecessors
A	Contact potential vendors	1	—
B	Review bid proposals	2	A
C	Locate financing	2	A
D	Select vendor	4	B
E	Negotiate contract	2	D
F	Prepare facilities	5	C
G	Install computer	2	E, F
H	Train operations manager	2	E, F
I	Implement application software	6	G, H
J	Train users	3	H
K	Validate installation	1	I, J

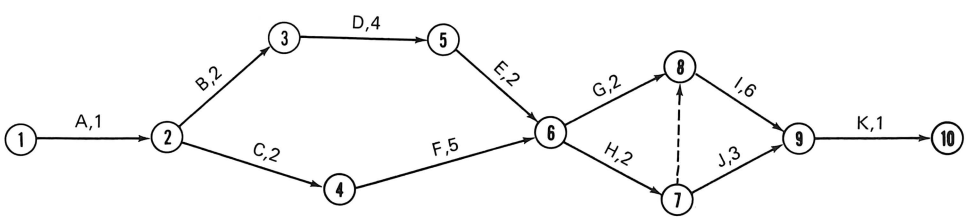


Figure 11.1—Acquire a Computer Project Network

nodes 6 and 8 because activity G is represented there. Only one activity arrow can be drawn between any two nodes.

One of the greatest benefits of using CPM is the resulting detailed planning. In order to draw the network, detailed planning is required. Drawing the network also brings out interactions between the activities, because, once the network is drawn, the entire project can be better visualized. In addition, because CPM is a formal technique understood by many people, communications are improved.

Having completed the CPM network, the next step is to calculate the minimum project duration. The first step in calculating this value is to determine the earliest time all activities ending at a node can be completed. This is done for all nodes making sure that precedence relationships are satisfied. Looking at Figure 11.2, we see that the earliest finish time for each node has been calculated; the times appear in the box by each node. The minimum time in which a computer can be acquired and implemented is 18 weeks. This number is obtained from the earliest finish time of the last node in the network. The

next step is to calculate the latest time each activity can be started without delaying the completion time (18 weeks) of the project. To do this we begin at the end of the network and work backward. The critical path through a CPM network is the sequence of activities that have no slack. We define slack for an activity to be:

Activity slack = latest start—earliest start, where

latest start = the latest time that an activity can start and not affect the project finish date

earliest start = the earliest time that an activity can start

One interpretation of slack is that an activity having slack can be delayed by this amount of time without delaying the completion time of the project. The activity slack appears in a triangle by each activity in Figure 14.2. Using these figures, we can list the activities on the critical path:

A — B — D — E — G — H — I — K

These activities all have a slack = 0.

11.2 CPM Program

Figure 11.3 contains a program that will determine the expected project duration and the critical path through a project network. As many as 100 activities can be in the network. This number can be changed by changing the value of IT at the beginning of the program.

Network data may be input from the keyboard or from a disk file. One restriction is that an activity beginning-node number must be less than the ending-node number. Another restriction is that there can only be one beginning (initial) node and one terminal node for a project network. This latter restriction may seem severe; however, by using dummy activities we can easily satisfy this restriction. Figure 11.4(a) contains the ending nodes of some project network. Only the ending two activities are shown. Figure 11.4(b) shows how we can use dummy activities (having no duration) to satisfy the requirement that the project can have only one ending node. In a similar manner the restriction of only one beginning node can be satisfied.

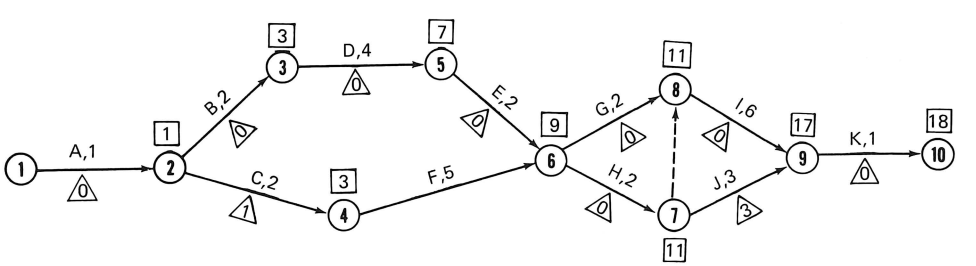


Figure 11.2—Acquire a Computer Project with CPM Computations

```
1000  REM       PROJECT PLANNING AND SCHEDULING PROGRAM: CPM
1010  REM       THIS PROGRAM USES THE CRITICAL PATH METHOD, CPM, TO
1020  REM       DETERMINE THE ESTIMATED COMPLETION TIME OF A PROJECT AND
1030  REM       THE ASSOCIATED CRITICAL PATH(S).  EACH ACTIVITY IN THE
1040  REM        PROJECT NETWORK IS REPRESENTED BY A BEGINNING NODE AND
1050  REM       AN ENDING NODE.  THE BEGINNING NODE NUMBER MUST BE LESS
1060  REM       THAN THE ENDING NODE NUMBER.  THE NETWORK CAN HAVE ONLY
1070  REM       ONE INITIAL NODE AND ONE TERMINAL NODE.  EACH ACTIVITY IS
1080  REM       IDENTIFIED BY AN ACTIVITY CODE.  A NETWORK MAY HAVE AS MANY
1090  REM       AS 100 ACTIVITIES.  LARGER NETWORKS MAY BE ANALYZED IF THE
1100  REM       PROGRAM DIMENSION STATEMENTS ARE CHANGED.  DATA MANAGEMENT
1110  REM       OPTIONS ARE PROVIDED THAT FACILITATE
1120  REM           -ADDING NEW ACTIVITIES
1130  REM           -CHANGING ACTIVITY DATA
1140  REM           -DELETING ACTIVITIES
1150  REM           -LISTING ACTIVITY DATA
1160  REM           -STORING NETWORK DATA
1170  REM           -IN A DISK FILE
1180 Z$ =  CHR$ (4): REM     CTRL-D
1190 IT = 100: REM      MAX. NO. ACTIVITIES
1200  DIM D$(IT),C$(IT),ST(IT),ED(IT),D(IT),AA(IT)
1210  DIM TT(2),TD$(2),EF(IT),IR(IT),ES(IT),XS(IT),XF(IT)
1220  REM      DEFINITION OF OUTPUT FORMATS
1230 D1$ = "ENTER ACTIVITY CODE, DESCRIPTION, BEG.  NODE, END NODE, DURATION"
1240 D2$ = "    ACTIVITY        BEGIN. END    ACTIVITY"
1250 D3$ = "CODE  DESCRIPTION    NODE  NODE  DURATION"
1260 D7$ = "ACT. DUR-  EARLY EARLY  LATE   LATE  SLK"
1270 D8$ = "CODE ATION START FINISH START FINISH TIM"
1280 D9$ = "    ACTIVITY          DUR-     EARLY    EARLY    LATE     LATE     SLACK"
1290 DO$ = "CODE  DESCRIPTION     ATION    START    FINISH   START    FINISH    TIME"
1300  HOME
1310  HTAB 15: PRINT "CPM PROGRAM": PRINT
1320  INPUT "DATA TO BE INPUT FROM KEYBOARD OR DISK  (K/D) ";Y$
1330  IF (Y$ <  > "K") AND (Y$ <  > "D") THEN  PRINT "INVALID ENTRY": GOTO
1340  IF Y$ = "K" THEN 1490
1350  PRINT : INPUT "ENTER NAME OF FILE ";N$
1360  PRINT Z$;"OPEN";N$
1370  PRINT Z$;"READ";N$
1380  INPUT T$: INPUT DT$: INPUT NAM$: INPUT IO
1390  FOR I = 1 TO IO
1400  INPUT C$(I): INPUT D$(I): INPUT ST(I): INPUT ED(I): INPUT D(I)
1410  NEXT I
1420  PRINT Z$;"CLOSE";N$
1430  PRINT : PRINT "DATE FROM DISK FILE IS: ";DT$: PRINT
1440  INPUT "DO YOU WANT TO CHANGE DATE (Y/N) ";Y$
1450  IF Y$ <  > "Y" THEN 1590
1460  PRINT : INPUT "ENTER NEW DATE ";DT$
1470  GOTO 1590
1480  PRINT : PRINT
1490  PRINT : INPUT "ENTER TITLE OF PROJECT ";T$
1500  PRINT
1510  INPUT "ENTER YOUR NAME ";NAM$
1520  PRINT
1530  INPUT "ENTER DATE ";DT$
1540  PRINT
1550  INPUT "ENTER TOTAL NUMBER OF ACTIVITIES ";IO: PRINT
1560  IF IO > IT THEN  PRINT "MUST BE <= ";IT: GOTO 1550
1570 II = 1
1580  GOSUB 3980: REM       NETWORK DATA INPUT
1590  GOSUB 4190: REM       SORT NETWORK ACT.
1600  REM  LIST PROGRAM OPTIONS
1610  PRINT : PRINT "PROGRAM OPTIONS"
1620  PRINT
1630  PRINT  TAB( 5)"1-ADD NEW ACTIVITIES"
1640  PRINT  TAB( 5)"2-CHANGE EXISTING ACTIVITY DATA"
1650  PRINT  TAB( 5)"3-DELETE ACTIVITIES FROM NETWORK"
```

```
1660  PRINT  TAB( 5)"4-LIST DATA AT CRT"
1670  PRINT  TAB( 5)"5-LIST DATA AT PRINTER"
1680  PRINT  TAB( 5)"6-STORE DATA ON DISK"
1690  PRINT  TAB( 5)"7-PERFORM CPM CALCULATIONS"
1700  PRINT  TAB( 5)"8-QUIT"
1710  PRINT
1720  INPUT "ENTER OPTION ";OPT: PRINT
1730  IF (OPT < 0) OR (OPT > 8) THEN  PRINT "INVALID": GOTO 1600
1740  IF OPT = 1 THEN IP = 1: GOTO 2010
1750  IF OPT = 2 THEN IP = 1: GOTO 2480
1760  IF OPT = 3 THEN IP = 1: GOTO 2110
1770  IF OPT = 4 THEN 1850
1780  IF OPT = 5 THEN 2720
1790  IF OPT = 6 THEN 3880
1800  IF OPT = 7 THEN 2900
1810  IF OPT = 8 THEN  PRINT  TAB( 10)"END OF PROGRAM": END
1820  GOTO 1600
1830  REM      ********************************************************
1840  REM      LIST DATA AT CRT
1850  HOME
1860  PRINT : PRINT "PROJECT: ";T$
1870  PRINT : PRINT "DATE: ";DT$: PRINT "NETWORK BY: ";NAM$: PRINT
1880  PRINT D2$
1890  PRINT D3$
1900 IC = 1
1910  FOR I = 1 TO IO
1920  IF I <  > (15 * IC) THEN 1960
1930 IC = IC + 1
1940  PRINT : INPUT "PRESS RETURN TO CONTINUE";Y$: PRINT
1950  PRINT D2$: PRINT D3$
1960  PRINT C$(I); TAB( 7)D$(I); TAB( 22)ST(I); TAB( 28)ED(I); TAB( 34)D(I)
1970  NEXT I
1980  PRINT : INPUT "PRESS RETURN TO CONTINUE ->";Y$: PRINT
1990  GOTO 1600
2000  REM      ********************************************************
2010  REM      ADD ACTIVITIES
2020  INPUT "ENTER NUMBER OF ACTIVITIES TO BE ADDED ";LA: PRINT
2030  IF LA < 0 THEN  PRINT "INVALID": GOTO 2020
2040  IF LA = 0 THEN 2870
2050  IF (LA + IO) > IT THEN  PRINT "MUST BE <= ";IT: GOTO 2020
2060 II = IO + 1
2070 IO = IO + LA
2080  GOSUB 3980
2090  GOTO 1600
2100  REM      ********************************************************
2110  REM      DELETE ACTIVITIES
2120  INPUT "ENTER NUMBER OF ACTIVITIES TO BE DELETED ";LA: PRINT
2130  IF LA < 0 THEN  PRINT "INVALID": GOTO 2120
2140  IF LA = 0 THEN 1600
2150  IF LA > IO THEN  PRINT "MUST BE <= ";IO: GOTO 2120
2160 II = 0
2170  FOR I = 1 TO LA
2180  INPUT "ENTER ACTIVITY CODE OF ACTIVITY TO BE DELETED ";AC$
2190  FOR L = 1 TO IO
2200  IF C$(L) <  > AC$ THEN 2250
2210 C$(L) = " ":D$(L) = " ":ST(L) = 0:ED(L) = 0:D(L) = 0
2220 II = II + 1
2230 IR(II) = L
2240  GOTO 2270
2250  NEXT L
2260  PRINT "NO MATCH FOUND FOR ACTIVITY CODE ";AC$
2270  NEXT I
2280  IF II = 0 THEN  GOTO 1600
2290  REM      MOVE ACTIVITY DATA INTO ARRAYS WHERE DATA WAS DELETED.
2300 LFT = IO - II
2310  FOR IL = LFT + 1 TO IO
```

```
2320  IF C$(IL) = " " THEN 2440
2330  IF IR(I) = IO THEN 2440
2340  FOR L = 1 TO II
2350  IF IR(L) > LFT THEN 2420
2360 LL = IR(L)
2370 C$(LL) = C$(IL):D$(LL) = D$(IL):ST(LL) = ST(IL)
2380 ED(LL) = ED(IL):D(LL) = D(IL)
2390 C$(IL) = " ":D$(IL) = " ":ST(IL) = 0:ED(IL) = 0:D(IL) = 0
2400 IR(L) = IO
2410  GOTO 2440
2420  NEXT L
2430  PRINT "COULD NOT FIND ANY SPACE IN DATA ARRAYS"
2440  NEXT IL
2450 IO = LFT
2460  GOTO 1600
2470  REM      ******************************************************
2480  REM      CHANGE ACTIVITY DATA
2490  INPUT "ENTER NUMBER OF ACTIVITIES TO BE CHANGED ";LA: PRINT
2500  IF LA < 0 THEN  PRINT "INVALID": GOTO 2490
2510  IF LA = 0 THEN 1600
2520  IF LA > IO THEN  PRINT "MUST BE <= ";IO: GOTO 2490
2530  FOR I = 1 TO LA
2540  PRINT : INPUT "ENTER ACTIVITY CODE OF ACTIVITY TO BE   CHANGED ";AC$: PRINT
2550  FOR L = 1 TO IO
2560  IF AC$ <  > C$(L) THEN 2660
2570  PRINT "CHANGES ARE FOR ACTIVITY ";L;" HAVING       ACTIVITY CODE ";C$(L)
2580  PRINT
2590  PRINT D1$
2600  PRINT : PRINT "ACTIVITY NO. ";L;
2610  INPUT C$(L),D$(L),ST(L),ED(L),D(L)
2620 C$(L) =  LEFT$ (C$(L),5):D$(L) =  LEFT$ (D$(L),14)
2630  IF ST(L) < ED(L) THEN 2680
2640  PRINT "BEGINNING NODE NO. MUST BE LESS THAN ENDING NODE NO."
2650  PRINT "REENTER DATA FOR THIS NODE": GOTO 2600
2660  NEXT L
2670  PRINT "NO MATCH FOUND FOR ACTIVITY CODE ";AC$
2680  NEXT I
2690  GOTO 1600
2700  REM      ******************************************************
2710  REM      LIST DATA AT PRINTER
2720  PRINT  CHR$ (4);"PR#1"
2730  PRINT : PRINT
2740  PRINT  TAB( 10)"PROJECT:";T$
2750  PRINT : PRINT "DATE: ";DT$; TAB( 40)"NETWORK BY: ";NAM$
2760  PRINT D2$
2770  PRINT D3$
2780  FOR I = 1 TO IO
2790  HTAB 1: PRINT C$(I);
2800  HTAB 7: PRINT D$(I);
2810  HTAB 21: PRINT ST(I);
2820  HTAB 27: PRINT ED(I);
2830  HTAB 33: PRINT D(I)
2840  NEXT I
2850  PRINT : PRINT
2860  PRINT  CHR$ (4);"PR#0"
2870  GOTO 1600
2880  REM      ******************************************************
2890  REM      CPM CALCULATIONS
2900  REM      PERFORM FORWARD PASS
2910  IF IP = 1 THEN  GOSUB 4190
2920 ES(1) = 0:EF(1) = D(1)
2930  FOR I = 2 TO IO
2940  IF ST(I) = ST(1) THEN ES(I) = 0:EF(I) = D(I): GOTO 3020
2950 MAX = 0
2960  FOR L = 1 TO IO
2970  IF ED(L) <  > ST(I) THEN 3000
```

```
2980  IF EF(L) > MAX THEN MAX = EF(L)
2990 ES(I) = MAX
3000  NEXT L
3010 EF(I) = ES(I) + D(I)
3020  NEXT I
3030  REM      PERFORM BACKWARD PASS
3040 LN = ED(IO)
3050 DX = 0
3060  FOR I = IO TO 1 STEP  - 1
3070  IF ED(I) <  > LN THEN 3100
3080  IF EF(I) > DX THEN DX = EF(I)
3090  NEXT I
3100 XF(IO) = DX
3110  FOR I = IO TO 1 STEP  - 1
3120  IF ED(I) = ED(IO) THEN XF(I) = DX:XS(I) = XF(I) - D(I): GOTO 3210
3130 MN = 999999
3140  FOR L = IO TO 1 STEP  - 1
3150  IF ST(L) < ED(I) THEN 3200
3160  IF ED(I) <  > ST(L) THEN 3190
3170  IF XS(L) < MN THEN MN = XS(L)
3180 XF(I) = MN
3190  NEXT L
3200 XS(I) = XF(I) - D(I)
3210  NEXT I
3220  REM       PERFORM SLACK CALCULATIONS
3230  FOR I = 1 TO IO
3240 AA(I) = XF(I) - EF(I)
3250  NEXT I
3260  INPUT "DISPLAY RESULTS ON CRT OR PRINTER (C/P)? ";Y$
3270  IF Y$ <  > "C" THEN 3440
3280  HOME
3290  PRINT : PRINT  TAB( 15)"CPM RESULTS": PRINT
3300  PRINT "PROJECT: ";T$
3310  PRINT "DATE: ";DT$: PRINT "NETWORK BY: ";NAM$: PRINT
3320 IC = 1
3330  PRINT D7$
3340  PRINT D8$
3350  FOR I = 1 TO IO
3360  IF I <  > (15 * IC) THEN 3390
3370 IC = IC + 1
3380  PRINT : INPUT "PRESS RETURN TO CONTINUE ->";Y$
3390  PRINT C$(I); TAB( 6)D(I); TAB( 12)ES(I); TAB( 18)EF(I); TAB( 25)XS(I); TAB(
     31)XF(I); TAB( 38)AA(I)
3400  NEXT I
3410  PRINT : PRINT "PROJECT DURATION:";DX: PRINT
3420  INPUT "LIST RESULTS ON THE PRINTER (Y/N) ";Y$
3430  IF Y$ <  > "Y" THEN 3840
3440  PRINT  CHR$ (4);"PR#1"
3450  PRINT  TAB( 25)"CPM RESULTS": PRINT : PRINT "PROJECT: ";T$
3460  PRINT "DATE: ";DT$; TAB( 30)"NETWORK BY: ";NAM$
3470  PRINT
3480  PRINT D9$
3490  PRINT D0$
3500  FOR I = 1 TO IO
3510 LV$ =  LEFT$ (C$(I),5)
3520  PRINT LV$;
3530 V1 = 6 -  LEN (LV$)
3540 LV$ =  LEFT$ (D$(I),14)
3550  PRINT  SPC( V1)LV$;
3560 V1 = 15 -  LEN (LV$)
3570 V2$ =  STR$ (D(I))
3580 LV$ =  LEFT$ (V2$,8)
3590 V2 =  LEN (LV$)
3600  PRINT  SPC( V1 + 8 - V2)V2$;
3610 V2$ =  STR$ (ES(I))
3620 LV$ =  LEFT$ (V2$,8)
```

```
3630 V1 = 9 -  LEN (LV$)
3640  PRINT  SPC( V1)LV$;
3650 V2$ =  STR$ (EF(I))
3660 LV$ =  LEFT$ (V2$,8)
3670 V1 = 8 -  LEN (LV$)
3680  PRINT  SPC( V1)LV$;
3690 V2$ =  STR$ (XS(I))
3700 LV$ =  LEFT$ (V2$,8)
3710 V1 = 8 -  LEN (LV$)
3720  PRINT  SPC( V1)LV$;
3730 V2$ =  STR$ (XF(I))
3740 LV$ =  LEFT$ (V2$,8)
3750 V1 = 9 -  LEN (LV$)
3760  PRINT  SPC( V1)LV$;
3770 V2$ =  STR$ (AA(I))
3780 LV$ =  LEFT$ (V2$,8)
3790 V1 = 7 -  LEN (LV$)
3800  PRINT  SPC( V1)LV$
3810  NEXT I
3820  PRINT : PRINT "PROJECT DURATION :";DX
3830  PRINT  CHR$ (4);"PR#0"
3840  PRINT : PRINT  TAB( 10)"END OF CPM ANAYSIS": FOR X = 1 TO 500: NEXT X
3850  GOTO 1600
3860  REM      ********************************************************
3870  REM      STORE NETWORK DATA ON DISK
3880  INPUT "ENTER NAME OF FILE ";N$
3890  PRINT Z$;"OPEN";N$
3900  PRINT Z$;"WRITE";N$
3910  PRINT T$: PRINT DT$: PRINT NAM$: PRINT IO
3920  FOR I = 1 TO IO
3930  PRINT C$(I): PRINT D$(I): PRINT ST(I): PRINT ED(I): PRINT D(I)
3940  NEXT I
3950  PRINT Z$;"CLOSE";N$
3960  GOTO 1600
3970  REM      ********************************************************
3980  REM      SUBROUTINE: NETWORK DATA INPUT
3990 IC = 1
4000 IM = 0
4010  PRINT D1$
4020  FOR I = II TO IO
4030 IM = IM + 1
4040  IF IM <  > (20 * IC) THEN 4080
4050 IC = IC + 1
4060  INPUT "PRESS RETURN TO CONTINUE ->";Y$
4070  PRINT D1$
4080  PRINT : PRINT "ACTIVITY NO. ";I;
4090  INPUT C$(I),D$(I),ST(I),ED(I),D(I)
4100 C$(I) =  LEFT$ (C$(I),5):D$(I) =  LEFT$ (D$(I),14)
4110  IF ST(I) < ED(I) THEN 4160
4120  PRINT
4130  PRINT "BEGINNING NODE NO. MUST BE LESS THAN ENDING NODE NO"
4140  PRINT
4150  PRINT "REENTER DATA FOR THIS ACTIVITY": GOTO 4080
4160  NEXT I
4170  PRINT : RETURN
4180  REM      ********************************************************
4190  REM      SUBROUTINE: SORT ACTIVITIES USING BEGINNING ACTIVITY NO.
4200  PRINT : PRINT "ACTIVITIES ARE BEING SORTED ": PRINT
4210 I1 = 1
4220  FOR I = I1 TO IO - 1
4230 IE = 0
4240  FOR L = I + 1 TO IO
4250  IF ST(I) < ST(L) THEN 4370
4260  IF ST(I) > ST(L) THEN 4290
4270  IF ED(I) < ED(L) THEN 4370
4280  IF ED(I) = ED(L) THEN 4390
```

```
4290 IM = L
4300 IE = 1
4310 TT(1) = ST(IM):TT(2) = ED(IM)
4320 TD$(1) = C$(IM):TD$(2) = D$(IM):TD = D(IM)
4330 ST(IM) = ST(I):ED(IM) = ED(I)
4340 C$(IM) = C$(I):D$(IM) = D$(I):D(IM) = D(I)
4350 ST(I) = TT(1):ED(I) = TT(2)
4360 C$(I) = TD$(1):D$(I) = TD$(2):D(I) = TD
4370  NEXT L
4380  GOTO 4410
4390  PRINT "SAME STARTING AND ENDING NODE NUMBERS-INVALID"
4400  PRINT "CHECK DATA FOR ACTIVITY CODES: ";C$(I),C$(L)
4410  NEXT I
4420  IF IE = 0 THEN 4450
4430 I1 = I1 + 1
4440  GOTO 4220
4450 IP = 0
4460  RETURN
```

Figure 11.3—CPM Program

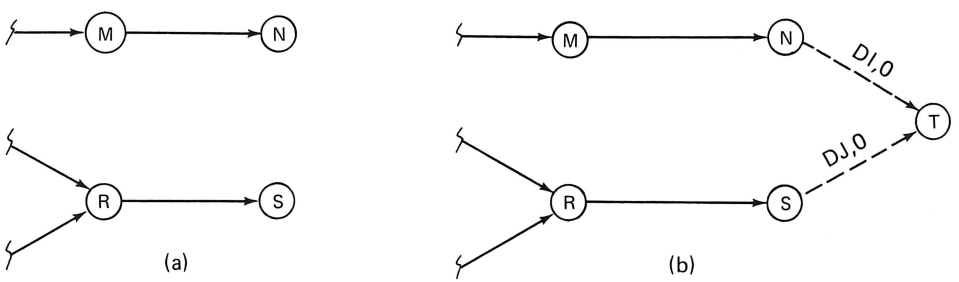


Figure 11.4—Satisfying the One Beginning Node Restriction

When data are input from the keyboard, the program prompts for the following information for each activity:

1. CODE
2. DESCRIPTION
3. BEGINNING NODE NUMBER
4. ENDING NODE NUMBER
5. DURATION

The activity code can be an alphanumeric value with five characters; the activity code should be unique (no other activity in the same network should have the same code). An activity description can have as many as fourteen alphanumeric characters. Node numbers and activity durations are limited by the report formats to no more than 7 characters including the decimal point.

If data are input from a disk file, the user is prompted for the file name. After the data are input from the keyboard or disk, the user must select an option from the following list:

1. ADD NEW ACTIVITIES
2. CHANGE EXISTING ACTIVITY DATA
3. DELETE ACTIVITIES FROM NETWORK
4. LIST DATA AT CRT
5. LIST DATA AT PRINTER
6. STORE DATA ON DISK
7. PERFORM CPM CALCULATIONS
8. QUIT

Several data management options are included in this list.

Option 1, ADD NEW ACTIVITIES, permits you to add additional activities to a network that is already in the computer. The program prompts you for the required information. Option 2, CHANGE EXISTING ACTIVITY DATA, lets you modify the data associated with an activity that is already in the computer. The program identifies all activities by the activity code; the activity description is only used to make the reports more meaningful. Option 3, DELETE ACTIVITIES FROM NETWORK, needs no explanation. Likewise, options 4 and 5, LIST DATA AT CRT and LIST DATA AT PRINTER, need no explanation. To avoid having to reenter your network from the keyboard, you can store your network in a disk file using option 6, STORE DATA ON DISK.

Once you are sure that your network is represented correctly in the computer, option 7, PERFORM CPM CALCULATIONS, can be executed. A report is output to the monitor containing the early start, early finish, late start, late finish, and slack for each activity. If you desire, a copy of this report will be listed at the printer.

11.3 Example CPM Problem

The data in Table 11.1 will be used to illustrate using the CPM program. The program output appears in **bold** characters, and user input appears in underlined characters.

EXAMPLE 1. Use the CPM program to determine the critical path through the network appearing in Figure 11.1.

SOLUTION. This program is stored in a file named CPM. Data can be input from the keyboard or from a disk file. Each time you run the program you are given an opportunity to specify the date the CPM analysis was performed. We will enter the activity information from the keyboard.

```
LOAD CPM

RUN

        CPM PROGRAM
```

```
DATA TO BE INPUT FROM KEYBOARD OR DISK
(K/D) K

ENTER TITLE OF PROJECT ACQUIRE A COMPUTER

ENTER YOUR NAME PHILIP WOLFE

ENTER DATE 6-5-83

ENTER TOTAL NUMBER OF ACTIVITIES 12

ENTER ACTIVITY CODE, DESCRIPTION, BEG.
NODE, END NODE, DURATION

ACTIVITY NO. 1?A,CONTACT VENDORS,1,2,1

ACTIVITY NO. 2?B,REVIEW PROPOSALS,2,3,2

ACTIVITY NO. 3?C,LOCATE FINANCING,2,4,2

ACTIVITY NO. 4?D,SELECT VENDOR,3,5,4

ACTIVITY NO. 5?E,NEGOTIATE CONTRACT,5,6,2

ACTIVITY NO. 6?F,PREPARE FACILITIES,4,6,5

ACTIVITY NO. 7?G,INSTALL COMPUTER,6,8,2

ACTIVITY NO. 8?H,TRAIN OP. MANAGER,6,7,2

ACTIVITY NO. 9?I,IMPLEMENT APPLICATIONS,8,9,6

ACTIVITY NO. 10?J,TRAIN USERS,7,9,3

ACTIVITY NO. 11?DUMI,DUMMY ACTIVITY,7,8,0

ACTIVITY NO. 12?K,VALIDATE INSTALLATION,9,10,1

ACTIVITIES BEING SORTED
```

After the activity data have been entered, the activities are sorted using the beginning node numbers. Because larger networks may take some time to sort,the program displays a message to let the user know what is occurring. Next we will store our network data in a disk file to avoid having to reenter the data if this network is studied at a later date. The data will be stored in a file named COMPUTER.

```
PROGRAM OPTIONS

   1-ADD NEW ACTIVITIES
   2-CHANGE EXISTING ACTIVITY DATA
   3-DELETE ACTIVITIES FROM NETWORK
   4-LIST DATA AT CRT
   5-LIST DATA AT PRINTER
   6-STORE DATA ON DISK
   7-PERFORM CPM CALCULATIONS
   8-QUIT
```

```
ENTER OPTION 6

ENTER NAME OF FILE COMPUTER

PROGRAM OPTIONS

     1-ADD NEW ACTIVITIES
     2-CHANGE EXISTING ACTIVITY DATA
     3-DELETE ACTIVITIES FROM NETWORK
     4-LIST DATA AT CRT
     5-LIST DATA AT PRINTER
     6-STORE DATA ON DISK
     7-PERFORM CPM CALCULATIONS
     8-QUIT
```

Before proceeding with the CPM analysis, we will display the network data at our CRT to see if the data are correct. Then, the CPM calculations will be performed.

```
ENTER OPTION 4

PROJECT:   ACQUIRE A COMPUTER

DATE:  6-5-83

NETWORK BY: PHILIP WOLFE
```

CODE	ACTIVITY DESCRIPTION	BEGINNING NODE NO	ENDING NODE NO	ACTIVITY DURATION
A	CONTACT VENDORS	1	2	1
B	REVIEW PROPOSA	2	3	2
C	LOCATE FINANCI	2	4	2
D	SELECT VENDOR	3	5	4
F	PREPARE FACILI	4	6	5
E	NEGOTIATE CONT	5	6	2
H	TRAIN OP. MANA	6	7	2
G	INSTALL COMPUT	6	8	2
DUM1	DUMMY ACTIVITY	7	8	0
J	TRAIN USERS	7	9	3
I	IMPLEMENT APPL	8	9	6
K	VALIDATE INSTA	9	10	1

```
PRESS RETURN TO CONTINUE

      PROGRAM OPTIONS
```

```
          1-ADD NEW ACTIVITIES
          2-CHANGE EXISTING ACTIVITY DATA
          3-DELETE ACTIVITIES FROM NETWORK
          4-LIST DATA AT CRT
          5-LIST DATA AT PRINTER
          6-STORE DATA ON DISK
          7-PERFORM CPM CALCUALTIONS
          8-QUIT

ENTER OPTION 7

DISPLAY RESULTS ON CRT OR PRINTER (C/P)?C

      CPM RESULTS

PROJECT:ACQUIRE A COMPUTER
DATE:  6-5-83
NETWORK BY: PHILIP WOLFE

ACT.     DUR-     EARLY     EARLY     LATE      LATE      SLK
CODE     ATION    START     FINISH    START     FINISH    TIME
A          1        0         1         0         1         0
B          2        1         3         1         3         0
C          2        1         3         2         4         1
D          4        3         7         3         7         0
F          5        3         8         4         9         0
E          2        7         9         7         9         1
H          2        9        11         9        11         0
G          2        9        11         9        11         0
DUMI       0       11        11        11        11         0
J          3       11        14        14        17         3
I          6       11        17        11        17         0
K          1       17        18        17        18         0

PROJECT DURATION:18

LIST RESULTS ON THE PRINER (Y/N)?N

       END OF CPM ANALYSIS
```

The results correspond to the project duration (18 weeks) and critical path depicted in Figure 14.2. It appears that we have a valid solution.

```
PRESS RETURN TO CONTINUE:

PROGRAM OPTIONS

     1-ADD NEW ACTIVITIES
     2-CHANGE EXISTING ACTIVITY DATA
     3-DELETE ACTIVITIES FROM NETWORK
     4-LIST DATA AT CRT
     5-LIST DATA AT PRINTER
     6-STORE DATA ON DISK
     7-PERFORM CPM CALCULATIONS
     8-QUIT
```

```
ENTER OPTION 8

          END OF PROGRAM
```

EXAMPLE 2. After some time had elapsed, management decided that another activity should be added to the Acquire A Computer project network. It was decided that activity K, VALIDATE INSTALLATION, would be deleted and replaced by the following activities:

CODE	DESCRIPTION	BEGINNING NODE	ENDING NODE	DURATION
L	AUDIT SYSTEM	9	10	2
M	VALIDATE INSTALLATION	10	11	1

Our task is to incorporate this new information into the project network and then perform the CPM analysis.

SOLUTION. The CPM program will be used to perform the required analysis. The task is relatively simple because we had stored the original network in a disk file named COMPUTER.

```
RUN

     CPM PROGRAM

DATA TO BE INPUT FROM KEYBAORD OR DISK
(K/D) D

ENTER NAME OF FILE COMPUTER

DATE FROM DISK FILE IS: 6-5-83

DO YOU WANT TO CHANGE DATE (Y/N) Y

ENTER NEW DATE 7-1-83

ACTIVITIES ARE BEING SORTED

     PROGRAM OPTIONS

        1-ADD NEW ACTIVITIES
        2-CHANGE EXISTING ACTIVITY DATA
        3-DELETE ACTIVITIES FROM NETWORK
        4-LIST DATA AT CRT
        5-LIST DATA AT PRINTER
        6-STORE DATA ON DISK
        7-PERFORM CPM CALCULATIONS
        8-QUIT
```

First we will delete activity K from the network, and then add activities L and M. After entering activities L and M, we note that the durations are incorrect. Therefore, these activities must be changed. After these changes are made we will display the data at the CRT to make sure that the data are correct. Next, the CPM calculations will be performed.

```
ENTER OPTION 3

ENTER NUMBER OF ACTIVITIES TO BE
DELETED 1

ENTER ACTIVITY CODE OF ACTIVITY TO BE
DELETED K

PROGRAM OPTIONS

     1-ADD NEW ACTIVITIES
     2-CHANGE EXISTING ACTIVITY DATA
     3-DELETE ACTIVITIES FROM NETWORK
     4-LIST DATA AT CRT
     5-LIST DATA AT PRINTER
     6-STORE DATA ON DISK
     7-PERFORM CPM CALCULATIONS
     8-QUIT

ENTER OPTION 1

ENTER NUMBER OF ACTIVITIES TO BE ADDED 2

ENTER ACTIVITY CODE, DESCRIPTION, BEG.
NODE, END NODE, DURATION

ACTIVITY NO. 12?L,AUDIT SYSTEM,9,10,1

ACTIVITY NO. 13?M,VALIDATE INSTALLATION,10,11,2

PROGRAM OPTIONS

     1-ADD NEW ACTIVITIES
     2-CHANGE EXISTING ACTIVITY DATA
     3-DELETE ACTIVITIES FROM NETWORK
     4-LIST DATA AT CRT
     5-LIST DATA AT PRINTER
     6-STORE DATA ON DISK
     7-PERFORM CPM CALCULATIONS
     8-QUIT

ENTER OPTION 2

ENTER NUMBER OF ACTIVITIES TO BE CHANGED 2

ENTER ACTIVITY CODE OF ACTIVITY TO BE
CHANGED L

CHANGES ARE FOR ACTIVITY 12 HAVING
ACTIVITY CODE L

ENTER ACTIVITY CODE, DESCRIPTION, BEG.
NODE, END NODE, DURATION

ACTIVITY NO. 12?L,AUDIT SYSTEM,9,10,2

ENTER ACTIVITY CODE OF ACTIVITY TO BE
CHANGED M
```

```
CHANGES ARE FOR ACTIVITY 13 HAVING
ACTIVITY CODE M

ENTER ACTIVITY CODE, DESCRIPTION, BEG.
NODE, END NODE, DURATION

ACTIVITY NO. 13?M,VALIDATE INSTALLATION,10,11,1

PROGRAM OPTIONS

     1-ADD NEW ACTIVITIES
     2-CHANGE EXISTING ACTIVITY DATA
     3-DELETE ACTIVITIES FROM NETWORK
     4-LIST DATA AT CRT
     5-LIST DATA AT PRINTER
     6-STORE DATA ON DISK
     7-PERFORM CPM CALCULATIONS
     8-QUIT

ENTER OPTION 4

PROJECT:   ACQUIRE A COMPUTER

DATE: 7-1-83
NETWORK BY: PHILIP WOLFE
```

CODE	ACTIVITY DESCRIPTION	BEGIN. NODE	END NODE	ACTIVITY DURATION
A	CONTACT VENDOR	1	2	1
B	REVIEW PROPOSA	2	3	2
C	LOCATE FINANCI	2	4	2
D	SELECT VENDOR	3	5	4
F	PREPARE FACILI	4	6	5
E	NEGOTIATE CONT	5	6	2
H	TRAIN OP. MANA	6	7	2
G	INSTALL COMPUT	6	8	2
DUM1	DUMMY ACTIVITY	7	8	0
J	TRAIN USERS	7	9	3
I	IMPLEMENT APPL	8	9	6
L	AUDIT SYSTEM	9	10	2
M	VALIDATE INSTA	10	11	1

```
PRESS RETURN TO CONTINUE

PROGRAM OPTIONS

      1-ADD NEW ACTIVITIES
      2-CHANGE EXISTING ACTIVITY DATA
      3-DELETE ACTIVITIES FROM NETWORK
      4-LIST DATA AT CRT
      5-LIST DATA AT PRINTER
      6-STORE DATA ON DISK
      7-PERFORM CPM CALCULATIONS
      8-QUIT

ENTER OPTION 7
```

```
ACTIVITIES ARE BEING SORTED

DISPLAY RESULTS ON CRT OR PRINTER (C/P)?C

        CPM RESULTS

PROJECT: ACQUIRE A COMPUTER
DATE: 7-1-83
NETWORK BY: PHILIP WOLFE

ACT.      DUR-      EARLY     EARLY     LATE      LATE      SLK
CODE      ATION     START     FINISH    START     FINISH    TIM
A           1         0         1         0         1        0
B           2         1         3         1         3        0
C           2         1         3         2         4        1
D           4         3         7         3         7        0
F           5         3         8         4         9        1
E           2         7         9         7         9        0
H           2         9        11         9        11        0
G           2         9        11         9        11        0
DUM1        0        11        11        11        11        0
J           3        11        14        14        17        3
I           6        11        17        11        17        0
L           2        17        19        17        19        0
M           1        19        20        19        20        0

PROJECT DURATION;20
```

The expected time to complete the revised project, twenty weeks, seems reasonable.

```
LIST RESULTS ON THE PRINTER (Y/N)? N

          END OF CPM ANALYSIS

PROGRAM OPTIONS

     1-ADD NEW ACTIVITIES
     2-CHANGE EXISTING ACTIVITY DATA
     3-DELETE ACTIVITIES FROM NETWORK
     4-LIST DATA AT CRT
     5-LIST DATA AT PRINTER
     6-STORE DATA ON DISK
     7-PERFORM CPM CALCUALTIONS
     8-QUIT

ENTER OPTION 8

                END OF PROGRAM
```

11.4 Summary

At some time most engineers and scientists will find themselves involved in project planning. As a result, a program was included in this book that implements the CPM project planning and scheduling technique. Space does not permit in-depth coverage of this topic. Books have been written on this subject covering such topics as cost/time

trade-offs and resource allocations. Weist (1977) contains a very good bibliography on this subject.

References

Moder, J. J., and C. R. Phillips, *Project Management with CPM and PERT*, Reinhold, New York, 1964.

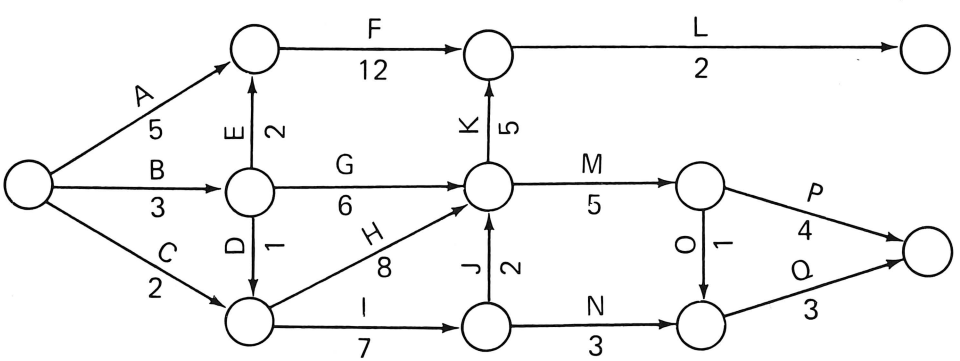
Weist, Jerome D., and Ferdinand K. Levy, *A Management Guide to PERT/CPM*, 2nd Edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1977.

Exercises

1. A project manager has asked for some assistance in determining the critical activities in a project. From the following data you are to:
   (a) draw a CPM network, and
   (b) determine the critical path and the expected project completion date.

Activity	Description	Duration Days	Immediate Predecessor
A	Activity 1	3	—
B	Activity 2	2	A
C	Activity 3	3	B
D	Activity 4	1	C
E	Activity 5	3	B
F	Activity 6	3	B
G	Activity 7	2	F,H
H	Activity 8	3	A
I	Activity 9	5	A
J	Activity 10	1	H
K	Activity 11	2	D,E,G,J
L	Activity 12	2	I
M	Activity 13	7	I
N	Activity 14	4	K,L,M

2. You are to find the critical path in the following network where the arrows represent activities and the numbers, by each arrow, represent duration in days.



3. A small construction project consists of 13 jobs. The precedence relationships are identified below in terms of node numbers. You are to:

a. Draw the network.
b. Determine the early and late start times for each job.
c. Identify the critical path.

Job	Initial Node, Final Node	Duration (Days)
J1	1, 2	3
J2	1, 3	1
J3	2, 4	5
J4	3, 4	2
J5	4, 5	4
J6	4, 6	3
J7	3, 6	2
J8	5, 9	2
J9	6, 7	1
J10	6, 8	3
J11	7, 8	1
J12	8,10	2
J13	9,10	1

4. A new research and develop project is being planned in a leading medical products manufacturing firm. The project network has been given to you in tabular form. You are to determine the expected duration of the project and the critical path.

Activity	Duration (Weeks)	Immediate Predecessor
A	2	—
B	4	A
C	3	A
D	2	A
E	2	C
F	1	B, E
G	2	C
H	4	C
I	5	C, D
J	2	F
K	6	G, N
L	4	F, K
M	1	G, N
N	2	H, I
O	7	H, I
P	3	O
Q	2	L, M
R	4	J, Q
S	2	J, Q
T	1	L, M, P
U	2	O
V	3	U
W	5	T
X	3	V
Y	2	S
Z	1	R, Y
AB2	2	W, X
MN4	2	AB2
DC5	2	Z
END	5	AB2, DC5

12

Sorting

One common task is the sorting of data according to some specified order. Arranging a list of numbers from smallest to largest or placing alphabetic information in alphabetical order are examples of sorting. This procedure can be a very time consuming task even when using a computer, especially if thousands of data elements are involved. As a result, several sorting procedures have been developed with the objective of making this task as efficient as possible. In this section, two sorting procedures will be presented.

12.1 Bubble Sort

The bubble sort is not very efficient, but it is easy to understand. Therefore, it is a good procedure to use as an introduction to sorting. Assume that you have a list of numbers that you want to sort into increasing order:

8 2 6 1 4

Using the bubble sort, the first two numbers are compared. If out of order, the positions of the numbers are swapped:

2 8 6 1 4

Then the numbers in positions two and three are compared, and if out of order, they are swapped:

2 6 8 1 4

This process continues until the numbers in the last two positions are compared and swapped if they are out of order:

2 6 1 8 4
2 6 1 4 8

If no swaps were made in the above process, the list is in sorted order. Note what has happened in this example; the largest number (8) has "bubbled" to the bottom of the list. At this point, the largest number is in the correct position; therefore, this number need not be considered in any future comparisons.

Another pass is made through the list. At the end of this pass, the list would be in the following order:

2 1 4 6 8

Now the bottom two numbers are in the correct positions.
The third pass results in:

1 2 4 6 8

The fourth pass results in no swaps; therefore, the list is in increasing order.

Bubble Sort Computer Program

A subroutine that will sort a list of numbers into ascending order using the bubble sort procedure appears in Figure 12.1. The list to be sorted must be in the array X and the number of entries in the list must be provided in the variable N.

```
23000  REM  SUBROUTINE: BUBBLE SORT
23010  REM  PROGRAM SORTS INTO INCREASING ORDER A ONE DIMENSIONAL
23020  REM  ARRAY, X, HAVING N NUMBERS.  THE ARRAY X AND ITS SIZE
23030  REM  N ARE SUPPLIED BY THE MAIN PROGRAM.  CHANGING THE >=
23040  REM  IN LINE 23090 TO <= WILL RESULT IN THE ARRAY BEING
23050  REM  SORTED IN DECREASING ORDER.
23060  FOR I = 1 TO N - 1
23070 IL = 0
23080  FOR J = 1 TO N - I
23090  IF X(J + 1) > = X(J) GOTO 23120
23100 A = X(J):X(J) = X(J + 1):X(J + 1) = A
23110 IL = 1
23120  NEXT J
23130  IF IL = 0 GOTO 23150
23140  NEXT I
23150  RETURN
```

Figure 12.1—Bubble Sort Program

EXAMPLE 1. Use the bubble sort procedure to sort the following list of numbers into ascending order:

38 21 7 32 2 45 13 60 18 55

SOLUTION. Figure 12.2 contains a program which was written to prompt the user for the list of numbers to be sorted. After the list is entered, the bubble sort subroutine is utilized to sort the numbers. The sorted list is then printed out.

```
100 DIM X(100)
110 INPUT "HOW MANY NUMBERS ARE TO BE SORTED";N
120 PRINT "ENTER";N;" DATA ELEMENTS"
130 FOR I=1 TO N
140 PRINT "ENTER ELEMENT";I;
150 INPUT X(I)
160 NEXT I
170 GOSUB 23000
180 FOR I=1 TO N
190 PRINT X(I)
200 NEXT I
210 END
```

Figure 12.2—Bubble Sort Application

12.2 Quicksort

The quicksort procedure will usually require less time than the bubble sort procedure to sort a randomly ordered list. However, if the list is already sorted or nearly sorted, the bubble sort can be faster.

The quicksort algorithm partitions the initial list into two lists with all the numbers in one subset being smaller than all of the numbers in the other subset. Then this procedure is repeated on each of the subsets, partitioning each of them into subsets. This process continues until each subset contains just one element. At this point the array is sorted.

This procedure is best explained with an example. Assume the list of numbers

7 2 14 10 6 4 9

is to be sorted into increasing order using the quicksort procedure. Initially, some number has to be chosen as a "pivot." This procedure will rearrange the above list into two subsets such that all of the numbers to the left are smaller than the pivot. All of the numbers to the right will be larger than the pivot.

For this example, the first number in the list will be chosen as the pivot.

```
P
7  2  14  10  6  4  9
```

The pivot element is then compared with the last number. If the last number is smaller than the pivot number, the numbers are swapped. Looking at our list, we see that 7 and 9 are in the proper order. The pivot is then compared to the next to last number, 4. Here the numbers are out of order, so the 7 and 4 are swapped:

```
                  P
4  2  14  10  5  7  9
   L              R
```

We will use an L and R to indicate which numbers at the left and right ends of the list have been compared with the pivot. Everything to the left of the L will have been compared with the pivot, and everything to the right of the R will have been compared with the pivot. Attention is now directed to the left end of the list, because we know that everything to the right of the number 7 is larger than 7 (at this point, only the number 9 is to the right of the pivot, 7).

Looking at the left end, we know that the last number exchanged with the pivot is less than the pivot because a swap was just made. Therefore, we start with the next number to the right, the number 2. Since 2 is less than the pivot, no swap occurs. However, 7 and 14 are out of order, so a swap occurs:

```
      P
4  2  7  10  6  14  9
      L      R
```

At this point, everything to the left of the pivot is less than 7 and everything from the number 14 to the right end is greater than 7. Therefore, the comparisons start with the first number to the left of the number 14. Another swap is made here because 6 is less than 7:

```
                    P
4   2   6   10   7   14   9
            L    R
```

Only the number 10 has not been considered. Again a swap is made:

```
                P
4   2   6   7   10   14   9
            L
            R
```

The list has now been partitioned into two subsets. Everything to the left of the pivot is less than the pivot, and everything to the right is greater than the pivot.

This same procedure is now applied to these subsets. First, consider the subset:

```
P
4   2   6
L       R
```

Select the first element (4) as the pivot. Since 6 and 4 are in the proper order, only one swap would be made:

```
    P
2   4   6
    L
    R
```

Everything to the left of the pivot (4) is less than the pivot and everything to the right is greater than the pivot. The result is two subsets containing one element each. Consequently, these numbers are in the proper order.

We have one remaining subset to sort:

```
P
10   14   9
L        R
```

Applying this procedure once again, we will select 10 as the pivot. Comparing the pivot with the rightmost element, we see that 10 is greater than 9, so these numbers must be swapped.

```
            P
9   14   10
    L    R
```

Then 10 and 14 are swapped

```
      P
9     10     14
      L
      R
```

Again we have partitioned a list into subsets of one number each. Our original list is now in increasing order

2 4 6 7 9 10 14

Consequently, the quicksort procedure has successfully sorted our original list.

The next step is to develop a computer program to implement the quicksort procedure. Comparing the various numbers is not difficult. However, developing a method for sorting the various subset of numbers to be partitioned requires some thought. We will start by establishing an arbitrary rule: when a set is partitioned, the right subset will be stored and the left subset will be partitioned. However, the right subset will only be saved to be partitioned later if it contains more than one element. Likewise, the left subset will only be partitioned if it contains more than one element.

A subset can be stored by saving the positions in the list of beginning and ending elements. These positions can be saved in two, one-dimensional arrays, one array containing the beginning position and one array containing the ending position. As the quicksort procedure is executed and subsets are created, the subsets on the right will be added to this list of subsets to be partitioned. When there are no more subsets on the left to be partitioned, the stored subsets will be removed from the one dimensional arrays in a last-in, first-out order (LIFO).

To illustrate this process, we will consider the following list:

7 2 14 10 6 4 9

The array LEF will be used to store the position of the first element in a subset and the array RIT will be used to store the position of the last element in a subset. Initially, we will have

LEF(1) = 1

RIT(1) = 7,

because the subset to be processed is the entire list, containing seven elements.

After partitioning the first subset, we obtain:

```
                P
4     2     6     7     10     14     9
```

Saving the right subset gives us

LEF(1) = 5

RIT(1) = 7

Remember, the LEF(1) and RIT(1) were not being used after the first subset, beginning in position 1 and ending in position 7, was removed to be partitioned.

Next, the left subset is partitioned, giving

P

2 4 6

Neither of the resulting subsets would be saved for additional processing since they contain only one element. Therefore, another subset must be removed from our subset array, that is the subset beginning in position 5 and ending in position 7. Thus, we get

10 14 9

Upon partitioning this subset, we get

P

9 10 14

Each of the resulting subsets contains only one element and our arrays LEF and RIT are empty. Consequently, the list is sorted in an increasing order. The procedure stops.

Refinements to the Quicksort Procedure

Before a program is introduced which uses the quicksort procedure, we will present some ideas which can improve the basic process. First, if the pivot element chosen is the largest or smallest element in the list, the result is a subset of one element and another subset of n-1 elements. Consequently, a better procedure for specifying the pivot element is to choose the median of the elements {X(1), X[(1 + n)/2], X(n)}, where X is an array holding the list to be sorted.

The amount of memory required to store the end positions of subsets remaining to be processed can be reduced. This can be accomplished by always storing the larger subset and immediately processing the smaller subset. Another improvement involves how the numbers being swapped are handled. Assume that a median value (as described above) is chosen as the pivot. This element can be removed from the array. If the first element is not chosen as the pivot element, the first element is moved to the empty position. The empty first position is then filled by comparing the elements in the list with the pivot element, starting from the right end of the list. The first element to be found less than the pivot element is moved into the empty position.

Consider our initial array:

7 2 14 10 6 4 9

The pivot element is selected from

X(1) = 7,

X[(1 + 7)/2] = X(4) = 10, and

X(7) = 9.

The median is X(7) = 9.

The pivot, 9, is removed from this list and the first element is moved to the empty position:

_ 2 14 10 6 4 7

Starting with the right end, elements are compared with the pivot element, 9, until one is found to be less than the pivot. In this case 7<9; therefore, the 7 is moved into the empty position:

7 2 14 10 6 4 _

This leaves the end position open; it can be filled by working from the left end until a larger element is found.

7 2 _ 10 6 4 14

Continuing this process results in:

7 2 4 10 6 _ 14

7 2 4 _ 6 10 14

7 2 4 6 _ 10 14

At this point, the list is partitioned into two subsets, and the pivot element is placed in the remaining empty position.

7 2 4 6 9 10 14

We are left with two subsets to be partitioned, one beginning in position 1 and ending in position 4, and another beginning in position 6 and ending in position 7. The largest subset will be stored to be partitioned later.

Since the small subset contains only two elements, we will choose the left element as the pivot. Comparing the elements, we see that no interchange is required; therefore, this subset is in the correct order.

EXAMPLE 2. Use the quicksort procedure to sort the list of numbers used in Example 1.

SOLUTION. The only difference between the solution to this example and the solution to Example 1 is that line 170 would be changed to

```
170 GOSUB 23200
```

specifying that the quicksort subroutine is to be used. Baase (1978) and Scheid (1982) provide additional material on sorting procedures.

```
23200  REM  SUBROUTINE: QUICKSORT
23210  REM  THIS SUBROUTINE USES THE QUICKSORT PROCEDURE TO SORT A LIST
23220  REM  OF NUMBERS INTO ASCENDING ORDER.  THE LIST MUST BE PROVIDED TO
23230  REM   THIS SUBROUTINE IN THE ARRAY X.  THE NUMBER OF ELEMENTS IN THE
23240  REM  LIST MUST BE SUPPLIED IN THE VARIABLE N.  THE ARRAYS LEF AND RIT
23250  REM  ARE USED TO STORE SUBSETS TO BE PARTITIONED.  LEF AND RIT
23260  REM  CONTAIN THE POSITION OF THE FIRST ELEMENT AND THE LAST ELEMENT
23270  REM  OF A SUBSET TO BE PARTITIONED.  LS AND RS MAINTAIN THE LEFT AND
23280  REM  RIGHT END POSITIONS OF THE SUBSET BEING PARTITIONED.  L AND R
23290  REM  INDICATE THE ELEMENTS THAT HAVE BEEN COMPARED WITH THE PIVOT,
23300  REM  PV.  PT INDICATES THE POSITION IN THE ARRAYS LEF AND RIT OF THE
23310  REM  LAST SUBSET STORED TO BE PARTITIONED.  IF ALL THE REM STATEMENTS
23320  REM  ARE REMOVED FROM WITHIN THE SORT PROCEDURE, THE SUBROUTINE WILL
23330  REM  SORT A LIST IN LESS TIME.
23340  REM  CHANGING THE > IN LINE 23710 TO < AND CHANGING THE < IN
23350  REM  LINE 23750 TO > WILL RESULT IN THE ARRAY BEING SORTED IN
23360  REM  DECREASING ORDER.
23370 SIZ% = 50
```

```
23380  DIM LEF(SIZ%),RIT(SIZ%)
23390 LEF%(1) = 1: REM  LEFT END OF SUBSET
23400 RIT%(1) = N: REM  RIGHT END OF SUBSET
23410  REM  POSITION IN LEF AND RIT OF LAST SUBSET STORED
23420 PT% = 1
23430  REM  SET LEFT SIDE POINTER
23440 L% = LEF%(PT%)
23450  REM  SAVE STARTING VALUE OF LEFT SIDE POINTER
23460 LS% = L%
23470  REM  SET RIGHT SIDE POINTER
23480 R% = RIT%(PT%)
23490  REM  SAVE STARTING VALUE OF RIGHT SIDE POINTER
23500 RS% = R%
23510 PT% = PT% - 1
23520  REM  IF ONLY 1 ELEMENT IN SUBSET, DO NOT PARTITION
23530  IF (R% - L%) <  = 0 THEN 24210
23540  REM  TEMPORARILY SET PIVOT TO LEFT ELEMENT
23550 PV = X(L%)
23560  REM  IF 5 OR LESS ELEMENTS IN SUBSET, LEAVE PIVOT AS IS
23570  IF (R% - L%) < 5 THEN 23700
23580  REM  SET PIVOT TO MEDIAN
23590 MED = (L% + R%) / 2
23600  IF (PV < X(R%)) AND (PV > X(MED)) THEN 23700
23610  IF (PV < X(MED)) AND (PV > X(R%)) THEN 23700
23620  IF (X(R%) < PV) AND (X(R%) > X(MED)) THEN 23680
23630  IF (X(R%) < X(MED)) AND (X(R%) > PV) THEN 23680
23640  REM  SET PIVOT=MEDIAN AND MOVE LEFT POINTER TO HOLE
23650 PV = X(MED)
23660 X(MED) = X(L%)
23670  GOTO 23700
23680 PV = X(R%)
23690 X(R%) = X(L%)
23700  IF (L% >  = R%) THEN 23840: REM  IF L=R, SUBSET PARTITIONED
23710  IF (PV > X(R%)) THEN 23780: REM  IF TRUE, SWAP
23720 R% = R% - 1: REM  MOVE POINTER
23730  GOTO 23700
23740  IF (L% >  = R%) THEN 23840: REM  IF L=R, SUBSET PARTITIONED
23750  IF (PV < X(L%)) THEN 23810: REM  IF TRUE, SWAP
23760 L% = L% + 1: REM  MOVE POINTER
23770  GOTO 23740
23780 X(L%) = X(R%): REM  PERFORM SWAP
23790 L% = L% + 1: REM  MOVE POINTER
23800  GOTO 23740
23810 X(R%) = X(L%): REM   PERFORM SWAP
23820 R% = R% - 1: REM  MOVE POINTER
23830  GOTO 23700
23840 X(L%) = PV: REM  PLACE PIVOT IN HOLE
23850  REM  DETERMINE WHICH SUBSET TO STORE; LARGEST IS TO BE STORED
23860  IF (L% - LS%) <  = (RS% - L%) THEN 24040
23870  REM  RIGHT SUBSET IS SMALLEST; IF IT CONTAINS MORE THAN 1
23880  REM  ELEMENT, STORE LEFT SUBSET
23890  IF (RS% - L%) <  = 1 THEN 23990
23900 LEF%(PT% + 1) = LS%
23910 RIT%(PT% + 1) = L% - 1
23920 L% = L% + 1
23930 LS% = L%
23940 R% = RS%
23950 PT% = PT% + 1
23960  GOTO 23530
23970  REM  RIGHT SUBSET CONTAINED ONLY 1 ELEMENT; PARTITION LEFT
23980  REM  SUBSET IF IT CONTAINS MORE THAN 1 ELEMENT
23990  IF (L% - LS%) <  = 1 THEN 24210
24000 R% = L% - 1
24010 RS% = L%
24020 L% = LS%
24030  GOTO 23530
```

```
24040  IF (L% - LS%) <  = 1 THEN 24140
24050  REM  LEFT SUBSET IS SMALLEST; IF IT CONTAINS MORE THAN 1
24060  REM  ELEMENT, STORE RIGHT SUBSET
24070 LEF%(PT% + 1) = L% + 1
24080 RIT%(PT% + 1) = RS%
24090 R% = L% - 1
24100 RS% = R%
24110 L% = LS%
24120 PT% = PT% + 1
24130  GOTO 23530
24140  IF (RS% - L%) <  = 1 THEN 24210
24150  REM  RIGHT SUBSET CONTAINED MORE THAN 1 ELEMENT; PARTITION IT
24160 L% = L% + 1
24170 LS% = L%
24180 R% = RS%
24190  GOTO 23530
24200  REM  ANY MORE SUBSETS TO BE PARTITIONED?
24210  IF (PT% < 1) THEN 24230
24220  GOTO 23440
24230  RETURN
```

Figure 12.3—Quicksort Program

12.3 Summary

Many computer applications require data to be sorted. Consequently, to make this text as useful as possible, two sort subroutines (bubble sort and quicksort) were presented. These subroutines can be used as provided.

References

Baase, Sara, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Massachusetts, 1978.

Scheid, Francis, *Computers and Programming*, McGraw-Hill, New York, 1982.

Exercises

1. Modify the bubble sort subroutine so that the array X is sorted into decreasing order. Test the resulting program using the data from example 1.
2. Modify the quicksort subroutine so that the array X is sorted into decreasing order. Test the resulting program using the data from example 1.
3. The following statements will generate 100 random integers in the range of $0 \leq x \leq 100$ (chapter 15 provides background for this program):

   ```
   10  DIM X(100)
   20  FOR I=1 TO 100
   30  X(I)=INT(100*RND(+1)+1)
   40  NEXT I
   ```

   Using these statements generate 100 random integers in the range of $1 \leq x \leq 100$. Next, sort these 100 numbers in increasing order using the bubble sort subroutine. Place a STOP statement in your program after the 100 random numbers are generated and before these numbers are sorted. Then, enter CONT from the keyboard and time how long it takes to sort the numbers.
4. Repeat exercises 3 using the quicksort subroutine.
5. Repeat exercises 3 and 4 using a list of 100 numbers that is already sorted before you attempt to sort them. Can you explain the results?

13

Disk Data Files

Your Apple microcomputer has the capability to support several types of memory storage devices. These devices can be grouped into two major types: primary memory and secondary memory. ROM (Read Only Memory) and RAM (Random Access Memory) are examples of primary storage devices. The Apple assembly language monitor resides in ROM; RAM is used to store your programs and data as the programs are executed. A cassette tape and a floppy diskette are examples of secondary storage devices.

When RAM devices are used to store information for long periods of time, they have a characteristic that can cause a problem. These devices are volatile; when power to a RAM device is turned off, the contents of the memory device are lost. One solution is to type in your program every time you turn on the computer; obviously, this is not an attractive one. A better alternative is to store the programs on a secondary storage medium, e.g., floppy disk, and load the program into RAM whenever you want to use it. Data can be stored on a secondary storage medium like a floppy disk in a similar manner; however, there are some simple concepts that must first be understood. These concepts will be discussed in this chapter. Note that devices other than diskettes can be used for secondary storage. This text, however, assumes that a disk will be used. Refer to your Apple reference manuals for details on using other secondary storage devices.

13.1 Recording Information on a Diskette

In order for a computer to read and write information to a diskette, some procedures must be established. Preferably, these procedures should be performed quickly and without error by the computer. They also should be easy for the user to understand and use.

As long as your programs and data are stored within the RAM memory of your computer, you will have little trouble identifying this information. However, when you store information on a diskette, it must be managed so that you can easily retrieve the appropriate information whether it is a program or data. Information is managed on secondary storage devices, such as a disk or tape, using the concept of a file. A *file* is a collection of related information that is kept somewhere other than in the primary memory of your personal computer.

For example, you might have a program that is used for curve fitting. This program can be saved on a diskette so that you could easily retrieve the program and use it again.

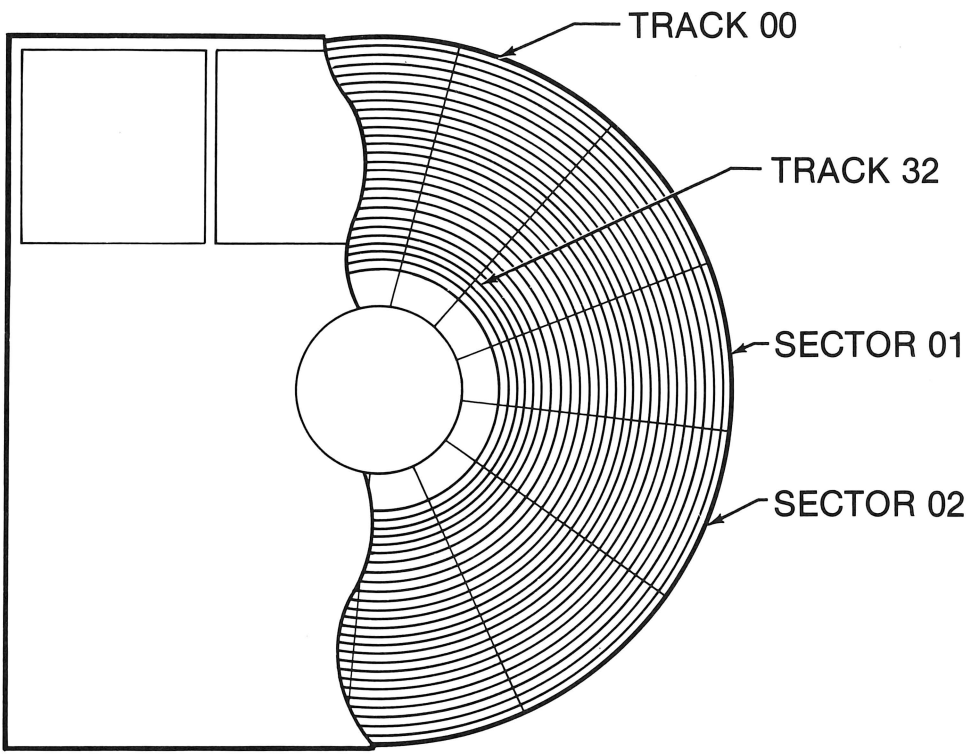


Figure 13.1—Floppy Disk Organization

In order to identify the program on the diskette you must assign a name to the file. One diskette can have as many as 105 files; therefore, a name provides an easy way to identify the various files.

Data files and program files can exist on the same diskette. However, each file on the diskette must have a unique name. The names of files on a particular diskette are maintained in the directory of that diskette. A file name must conform to the following rules:

1. The name may be 1 to 30 characters long.
2. The first character of the name must be a letter (A–Z).
3. Any character on the keyboard, except a comma, may be part of the name.

Here are some valid names for a diskette file:

ANALYSIS.BAS
DESIGN
DESIGN.JAN
TESTA 82.JAN

RESULT1

If your system has more than one disk drive, specifying a file name may not be enough for the computer to locate the file. In addition, you may need to specify the disk drive containing the diskette on which the specified file is stored. Therefore, the complete file specification is

filename,Dn

where the Dn specifies that the file is located on a diskette in disk drive n. After you specify a particular disk drive, subsequent disk commands default to that drive unless another drive is specified. A default disk drive is the drive that DOS assumes will contain the diskette or file you specify in a command unless another drive is specified. Some examples are:

```
LOAD DESIGN,D2
SAVE RESULT1
```

In these examples, the file DESIGN was LOADed into memory from disk drive two. The second command SAVEs a copy of the program in memory in a file called RESULT1 on disk drive two, which is the default drive at this point.

Figure 13.1 illustrates how information is organized on a diskette. Each diskette contains 35 concentric tracks on one side. Each of these tracks is divided into smaller segments, called sectors. There are 16 sectors per track, and 256 bytes may be stored in each sector.

Theoretically, your Apple microcomputer can store 143,360 bytes on one side of a diskette, assuming that you are using DOS 3.3 as earlier versions of DOS did not store this much information on a diskette. This value is obtained as follows: 35 tracks × 16 sectors/track × 256 bytes/sector = 143,360 bytes. However, not all of this space is available for data storage, because the disk operating system (DOS) must use some of the space to manage the information stored on a diskette.

After space is allocated for DOS usage, there are 496 sectors (12,976 bytes) available for data. The information maintained by DOS on each diskette includes a track/sector list and a directory. One primary function of the track/sector list is used by DOS to determine where on the diskette the data are stored.

The primary function of the DOS diskette directory is to maintain a table containing the names of files that have been saved on the diskette. This table also contains other information such as the file type and the number of sectors it uses. There is room in the directory for 105 file names. As a result, 105 is the maximum number of files that can be saved on a diskette even though not all of the sectors have been used. If you try to save more than 105 files, you will receive an error message. For additional information regarding the track/sector list and the diskette directory, refer to your *DOS Manual (DOS 3.3)*.

13.2 Sequential Access Files

There are two types of diskette data files that can be used with Apple DOS: sequential and random access. Sequential access methods are easier to learn; however, you will find that this type of (input/output) I/O is limited in flexibility and speed. These limitations will be discussed after random access is explained. The sequential access method assumes

that data are written to a file one entry after another. If you want to retrieve the tenth entry, you must retrieve the first nine entries before the tenth entry can be retrieved. You might visualize a sequential file as residing on a tape similar to a tape on your home stereo. Before you can play a song (a specific file), you must position the tape at the beginning at the desired song (file). If you want to play the third song, you must first pass the first two songs on the tape. This is an example of sequential access.

Apple DOS uses a temporary storage area (buffer) to manage data that are written to and read from a disk. Sequential access requires a buffer area for writing to the disk and another buffer area for reading from the disk. Figure 13.2 illustrates these buffer areas. For sequential access, the write and read buffers are each fixed at 256 bytes. These buffers permit BASIC to be more efficient in accessing information on a disk. Information to be written to the disk is temporarily stored in the buffer until the buffer is full. Then this information is transferred by DOS to the diskette. Each disk access is relatively slow because mechanical devices are involved. Therefore, using a buffer in this manner requires fewer disk accesses resulting in improved program execution time. A buffer is used in a similar manner whenever a read operation is performed.

Writing and Reading Sequential Files

The following steps are required to write data to a sequential file, and then read those data from the file:

Write to a Sequential File

1. Specify file name.
2. Specify write or append mode.
3. Write data to the file.
4. Close file.

Read from a Sequential

1. Specify file name.
2. Specify read mode.
3. Read data from file.
4. Close file.

In apple DOS, the first of the above steps is performed with one of the following DOS commands: OPEN or APPEND. If a DOS command, such as OPEN, is used within a program, the command must be in a PRINT statement and be preceded by a CTRL-D character (hold the CTRL key down while you press the D key). For example, to OPEN the file DESIGN, residing on disk drive two, we can use the following statements:

```
10 D$=CHR$(4): REM CTRL-D
20 PRINT D$; "OPEN DESIGN,D2"
```

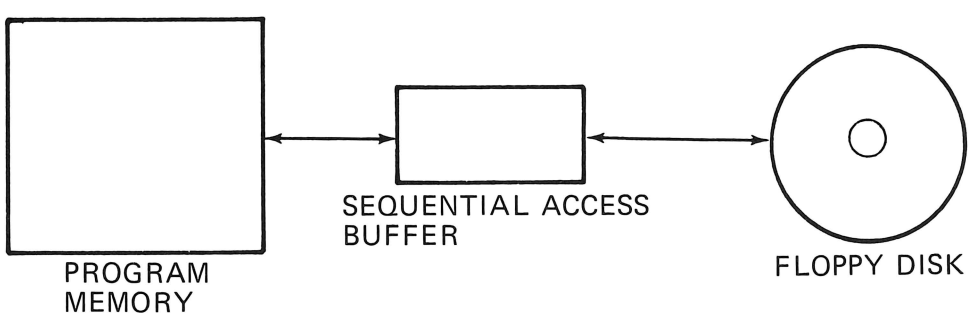


Figure 13.2—Illustration of Sequential Access Buffer Area

After these statements are executed, a buffer area in memory will have been reserved for the file DESIGN. This buffer will hold one sector (256 bytes) of information before it is written to the disk. Information will not be written to the disk until this buffer is full or until a CLOSE command is executed. The above statements also enter the name DESIGN in the directory of the diskette in drive two (if this file did not already exist). In addition, an internal pointer is set to the beginning of the file; this pointer indicates the next location in the file where information is written or read.

Apple DOS provides three modes for accessing a sequential disk file: WRITE (output), APPEND (output), and READ (input). Information is written to a disk using a PRINT statement, the same statement is used to write to a CRT or a printer. Therefore, before you try to write to a disk, you must first specify where the information is to be PRINTed. This can be done as follows:

```
30 PRINT D$;"WRITE DESIGN"
```

After executing this statement, subsequent output (PRINT specifications) will be stored in the sequential disk file DESIGN. Even error messages will be sent to this disk file. As a result, after an error message is stored, the DOS write command is cancelled and control is returned to DOS. At any time you can cancel the WRITE command by specifying any DOS command or by using the null command;

```
100 PRINT D$
```

When this command is executed, subsequent output will appear at the screen or printer.

We noted above that the OPEN statement sets the internal pointer to the beginning of a file. However, there are times when we may want to add information to the end of a file. To do this, we must first locate the end of the file; the APPEND command will set the pointer to the end of the file. Thus, to APPEND information to the file DESIGN, execute the following command:

```
20 PRINT D$; "APPEND DESIGN,D2"
```

You would use this command in place of the OPEN command. It also establishes a buffer area in memory, but this command will not create a file named DESIGN in the diskette directory. Consequently, if this file does not already exist, the error message

```
FILE NOT FOUND
```

will appear on the screen.

OPENing a sequential file sets the internal pointer to the beginning of a file, but any existing data are not destroyed. Therefore, subsequent PRINT statements will write over the previous data. If fewer characters than were initially in the file are output, some of the old data will remain at the end of the file. So, a good rule to follow when recreating an existing file is to first delete the old file:

```
20 PRINT D$; "OPEN DESIGN,D2"
25 PRINT D$; "DELETE DESIGN"
30 PRINT D$; "OPEN DESIGN"
```

The above statements will work even if the file DESIGN does not initially exist because statement 20 will create a file by this name if it does not exist. Statement 30 creates a new empty file called DESIGN.

Closing a file lets you change the access mode. If the file had been open for WRITE or APPEND, closing the file causes the final buffer contents to be written to disk. If a file will not be used continuously, it is a good practice to close it. You thus insure that should a power failure occur, the contents of the output buffer will not be lost. You may CLOSE all open files with the following command:

```
40 PRINT D$; "CLOSE"
```

Or you may close a specific file by adding the file name to the CLOSE statement:

```
40 PRINT D$; "CLOSE FILENAME"
```

After you have opened a file for output, you may write data to the file using the PRINT statement. For example, assume that you want to output to a file the string variable "EXPERIMENT$" and the numeric variable "VALUE." This could be done using the instructions:

```
10 D$=CHR$(4): REM CTRL-D
20 PRINT D$; "OPEN DESIGN,D2"
30 PRINT D$; "WRITE DESIGN"
40 PRINT EXPERIMENT$;CHR$(13);VALUE
```

Note that something new has been added to line 40, CHR$(13), which is the ASCII code for carriage return. Apple DOS requires that each field in a sequential file be separated by a carriage return. A carriage return is automatically inserted after the last field in a PRINT statement. However, when several fields are contained in one statement, you must provide the necessary separating carriage returns. An alternate way to write statement 40 would be

```
40 PRINT EXPERIMENT$: PRINT VALUE
```

You can remember what is required to print (write) information in a data file if you remember that the PRINT statement writes data to a file as it would be displayed on the screen. For instance, let X$="EXAMPLE" and Y$="ONE". Using the statement

```
40 PRINT X$;Y$
```

the following characters would be written in the file:

```
          C
EXAMPLEONER
          C
```

The symbols R represent carriage return. This is the same image that would have been displayed on your monitor if a PRINT statement had been used (the carriage return symbols are not displayed). The problem occurs when you try to input this image from the file. Because there are no delimiters between the two words, the computer assumes there is only one word. This problem can be avoided by including delimiters in the PRINT statement such as:

```
40 PRINT X$;CHR$(13);Y$
```

Executing this statement will result in the following image being written to the file:

```
        C   C
EXAMPLERONER
```

Therefore, when you use a PRINT statement to write to a file, care must be taken to properly delimit the data on the file.

EXAMPLE 1. Create a sequential data file consisting of date and high temperature for each day during the month of January. Let DAY$ denote the date and TEMP denote the high temperature for that date. The program should continue asking for data until you type "END" for the date.

SOLUTION.

```
 10 D$=CHR$(4): REM CTRL-D
 20 PRINT D$; "OPEN HITEMPJAN"
 30 INPUT "DATE ?";DAY$
 40 IF DAY$="END" GOTO 100
 50 INPUT "TEMPERATURE ?";TEMP
 60 PRINT D$; "WRITE HITEMPJAN"
 70 PRINT DAY$:PRINT TEMP
 80 PRINT D$
 90 GOTO 30
100 PRINT D$; "CLOSE"
110 END
```

Note that statement 80 is required to cancel the DOS write command so that subsequent output will appear at the CRT.

Assume that when this program is run, the following data are input:

```
1/3/83
33.56
1/4/83
18.32
END
```

After writing this information to the diskette, the file ("HITEMPJAN") would contain the following information:

```
     C       C       C       C
1/383R33.56R1/4/83R18.32R
```

In the above example, PRINT statements, involving a disk write operation (statement 70), were executed four times; consequently, there are four carriage return entries in the file. All information in a sequential file is stored as ASCII characters. Even a number such as 33.56 is stored as five ASCII characters rather than the binary number representation of the numeric value.

EXAMPLE 2. Write a program that will search the data file created in Example 1 for the date 1/4/83 and print on the screen the temperature for this date. If the specified date is not found, the program should print a message indicating this condition.

SOLUTION.

```
200 ONERR GOTO 310: REM SET ERROR TRAP
210 D$=CHR$(4): REM CTRL-D
220 INPUT "DATE TO SEARCH FOR ?";MACH$
230 PRINT D$; "OPEN HITEMPJAN"
240 PRINT D$; "READ HITEMPJAN"
250 INPUT DAY$,TEMP
```

```
260 IF MACH$=DAY$ THEN GOTO 290
270 GOTO 250
275 REM MATCH FOUND
280 POKE 216,0: REM CANCEL ERROR TRAP
290 PRINT "DATE :";DAY$,"HIGH TEMP :";TEMP
300 GOTO 330
305 REM NO MATCH FOUND
310 POKE 216,0: REM CANCEL ERROR TRAP
320 PRINT "NO MATCH FOUND"
330 PRINT D$; "CLOSE"
340 END
```

If a program tries to read beyond the last data element in a file, an error will be generated and the program will abort. Errors can be "trapped" using the ONERR GOTO statement. This means that you can prevent the program from printing an error message and aborting when an error occurs. You may want to do this because the error provides useful information. For example, in the above program, and END OF DATA condition means that the desired date was not found in the file. To cancel the error trapping condition, use the statement POKE 216,0.

Some applications may involve large amounts of data. Consequently, you need to know how much data you can store on a diskette. Each sector on a diskette can store 256 bytes of information. Looking at Example 1 above, we can determine how many days of temperature data will fit into one sector. For each day, the following information is written to the diskette:

```
      C       C
1/3/83R33.56R
```

Counting characters, we see that 13 characters are required. In a sequential file, each character requires 1 byte. Assuming that each day requires 13 characters, data for 19 days can be stored in 1 sector (256/13 = 19.69). Space on a diskette is allocated in blocks of 256 bytes (one sector). If a file uses less than a full sector, the remaining space is not available to be assigned to other files. Consequently, this unused space is wasted.

SEQUENTIAL FILE MANAGEMENT PROGRAM

It is difficult, if not impossible, to write a sequential file management program that can be used for all sequential access applications. Also, programming is largely an art; therefore, we learn from seeing how others handle a given problem. As a result, the approach used here is to present an example that illustrates many of the concepts we have been discussing as well as some new ideas.

EXAMPLE 3. All work received by Speciality Engineering Co. is obtained by submitting a bid to prospective clients. At any time, this company may have 80 to 100 jobs in one of the following stages:

1. Working on bid.
2. Waiting to start work.
3. Work in process.
4. Complete.
5. Past due.

Write a program that will maintain a job inventory in a sequential disk file. The following information should be maintained for each bid:

1. Job number.
2. Job description.
3. Job due date.
4. Bid dollar amount.
5. Status (bidding, complete, late, etc.)

Include in the program the ability to create a sequential file, append to the file, modify the file contents, and list the file contents.

SOLUTION. Interactive computer programs often are developed using a "menu" consisting of different options. When executing the program, the user must specify which option is to be performed. This concept was used to develop this program.

A menu is provided consisting of five options:

1—Create (Create a new job inventory data file)
2—Append (Add new jobs to this file)
3—Modify (Modify job information in this file)
4—List (List job inventory data)
5—Quit (Exit from program)

A subroutine is used for each option except the last, QUIT. A listing of this program appears in Figure 13.3.

Option 1, CREATE, will create a file having the name specified by the user at the beginning of the program. Before a file is created, the program determines if the file name specified is unique, because, creating (and writing) this file will destroy the contents of a file having the same name. If a file with the same name is found, the user is given the option of aborting or continuing with the creation of this new file. If the file is created, the program asks for data. The user denotes end of data by typing "END" for the JOB NO., and the data file is closed and control is returned to the menu.

You might note that an ONERR GOTO statement is used in the the CREATE subroutine. To determine if a file name is unused, we first try to append to a file by that name. If the file does not exist, Apple DOS returns the error message.

```
FILE NOT FOUND
```

This is error code number 6; you can find the error code in memory location 222 using PEEK(222). When this error occurs, program execution is terminated unless the error is "trapped." The ONERR statement permits us to trap the error and provides a means for dealing with it. In this case, the occurrence of the error is good because it means that no file exists having the name we want to use. The statement

```
POKE 216,0
```

disables error trapping.

The second option, APPEND, permits us to add data to our file whenever we desire. Like option 1, this option terminates whenever "END" is entered for a JOB NO. The data file is then closed and control is returned to the menu.

```
1000  REM  PROGRAM:SEQMAN
1010  REM  EXAMPLE SEQUENTIAL FILE DATA MANAGEMENT PROGRAM
1020  REM  MAINTAINS A JOB INVENTORY USING THE SEQUENTIAL ACCESS METHOD.
1030  REM  THE FOLLOWING INFORMATION IS STORED FOR EACH JOB:
1040  REM                1-JOB NUMBER
1050  REM                2-JOB DESCRIPTION
1060  REM                3-DUE DATE
1070  REM                4-BID DOLLAR AMOUNT
1080  REM                5-JOB STATUS
1090  REM
1100 DOS$ =  CHR$ (4): REM  CONTROL-D
1110  PRINT : PRINT "ENTER FILE NAME TO BE USED FOR JOB"
1120  INPUT "INVENTORY ? ";NAM$
1130  PRINT : PRINT  TAB( 5)"MENU:"
1140  PRINT  TAB( 10)"1- CREATE"
1150  PRINT  TAB( 10)"2- APPEND"
1160  PRINT  TAB( 10)"3- MODIFY"
1170  PRINT  TAB( 10)"4- LIST"
1180  PRINT  TAB( 10)"5- QUIT"
1190  PRINT : INPUT "OPTION ? ";OPT%
1200  IF (OPT% < 1) OR (OPT% > 5) GOTO 1230
1210  ON OPT% GOSUB 1270,1420,1880,1630,1240
1220  GOTO 1130
1230  PRINT : PRINT "INVALID OPTION, TRY AGAIN!": GOTO 1190
1240  PRINT : PRINT  TAB( 11);"*** END OF PROGRAM ***"
1250  PRINT DOS$;"CLOSE": END
1260  REM  ************************************************************
1270  REM  SUBROUTINE: CREATE A SEQUENTIAL FILE AND ACCEPT DATA
1280  REM  DETERMINE IF FILE NAME IS UNIQUE
1290  ONERR  GOTO 1360
1300  PRINT DOS$;"APPEND ";NAM$
1310  PRINT DOS$;"CLOSE ";NAM$
1320  PRINT : PRINT "FILE ALREADY EXISTS WITH THIS NAME: ";NAM$
1330  INPUT "CREATING NEW FILE WILL DESTROY CONTENTS OF OLD FILE  ABORT (Y/N) ? ";AN$
1340  IF AN$ = "Y" THEN  RETURN
1350  GOTO 1390
1360 Y =  PEEK (222): IF Y = 6 THEN 1380
1370  RESUME
1380  POKE 216,0
1390  PRINT DOS$;"OPEN ";NAM$
1400  GOTO 1450
1410  REM  ************************************************************
1420  REM  SUBROUTINE: APPEND TO FILE
1430  PRINT DOS$;"APPEND ";NAM$
1440  REM  REQUEST DATA
1450  PRINT : INPUT "ENTER JOB NO.(END TO STOP) ? ";JNO$
1460  IF JNO$ = "" THEN 1450
1470  IF  LEN (JNO$) > 3 THEN  PRINT : PRINT "ONLY 3 DIGITS ALLOWED": GOTO 1450
1480  REM  END OF INPUT?
1490  IF JNO$ = "END" THEN 1610
1500  PRINT : INPUT "ENTER JOB DESCRIPTION ? ";DESC$
1510 DESC$ =  LEFT$ (DESC$,13)
1520  PRINT : INPUT "ENTER DUE DATE ? ";DA$
1530 DA$ =  LEFT$ (DA$,8)
1540  PRINT : INPUT "ENTER BID $ AMOUNT (UP TO 8 DIGITS) ?   $ ";AMOUNT
1550  PRINT : INPUT "ENTER JOB STATUS ? ";ST$
1560 ST$ =  LEFT$ (ST$,3)
1570  PRINT DOS$;"WRITE ";NAM$
1580  PRINT JNO$: PRINT DESC$: PRINT DA$: PRINT AMOUNT: PRINT ST$
1590  PRINT DOS$
1600  GOTO 1450
1610  PRINT DOS$;"CLOSE";NAM$: GOTO 1130
1620  REM  ************************************************************
1630  REM  SUBROUTINE: LIST DATA
1640  REM  OPEN FILE FOR INPUT
1650  GOSUB 2410
```

```
1660  REM  ON EOF STOP READING DATA
1670  ONERR  GOTO 1830
1680 IC% =.1
1690 LINENO% = 1
1700  PRINT "JOB"; TAB( 21);"DUE"; TAB( 31);"$"
1710  PRINT " NO."; TAB( 6);"DESCRIPTION"; TAB( 21);"DATE"; TAB( 29);"AMOUNT"; TAB( 36);"STAT"
1720  PRINT "---------------------------------------"
1730 LINENO% = LINENO% + 1
1740  IF LINENO% <  > (IC% * 15) THEN 1780
1750  REM  STOP EXECUTION TO READ DATA
1760 IC% = IC% + 1
1770  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
1780  PRINT DOS$;"READ ";NAM$
1790  INPUT JNO$: INPUT DESC$: INPUT DA$: INPUT AMOUNT: INPUT ST$
1800  PRINT DOS$
1810  PRINT  TAB( 1);JNO$; TAB( 5);DESC$; TAB( 19);DA$; TAB( 29);AMOUNT; TAB( 37);ST$
1820  GOTO 1730
1830  POKE 216,0: PRINT DOS$;"CLOSE";NAM$
1840  PRINT : PRINT "END OF FILE": PRINT
1850  INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
1860  GOTO 1130
1870  REM  ************************************************************
1880  REM  SUBROUTINE: MODIFY DATA
1890  REM  OPEN FILE FOR INPUT
1900  GOSUB 2410
1910  PRINT : INPUT "ENTER JOB TO BE MODIFIED ? ";JNO$
1920  REM  FIND SPECIFIED DATA ID FOR MODIFICATION
1930  REM  OPEN SCRATCH FILE
1940  PRINT DOS$;"OPEN SCRATCH"
1950  ONERR  GOTO 2030
1960  PRINT DOS$;"READ ";NAM$
1970  INPUT TJNO$: INPUT TDESC$: INPUT FDA$: INPUT TAMOUNT: INPUT TST$
1980  IF TJNO$ = JNO$ THEN 2110
1990  REM  WRITE TO SCRATCH FILE
2000  PRINT DOS$;"WRITE SCRATCH"
2010  PRINT TJNO$: PRINT TDESC$: PRINT FDA$: PRINT TAMOUNT: PRINT TST$
2020  GOTO 1960
2030  PRINT : PRINT "END OF FILE-NO MATCH FOUND FOR: ";JNO$
2040  POKE 216,0
2050  PRINT DOS$;"CLOSE SCRATCH"
2060  REM  ERASE SCRATCH FILE BECAUSE MATCH NOT FOUND
2070  PRINT DOS$;"DELETE SCRATCH"
2080  PRINT DOS$;"CLOSE ";NAM$
2090  GOTO 1130
2100  REM  MODIFY DATA FOR SPECIFIED JOB
2110  POKE 216,0: PRINT DOS$
2120  PRINT : INPUT "ENTER JOB NO. ? ";JNO$
2130  IF  LEN (JNO$) > 3 THEN  PRINT : PRINT "ONLY 3 DIGITS ALLOWED": GOTO 2120
2140  PRINT : INPUT "ENTER JOB DESCRIPTION ? ";DESC$
2150 DESC$ =  LEFT$ (DESC$,13)
2160  PRINT : INPUT "ENTER DUE DATE ? ";DA$
2170 DA$ =  LEFT$ (DA$,8)
2180  PRINT : INPUT "ENTER BID $ AMOUNT (UP TO 8 DIGITS) ?   $ ";AMOUNT
2190  PRINT : INPUT "ENTER JOB STATUS ? ";ST$
2200 ST$ =  LEFT$ (ST$,3)
2210  ONERR  GOTO 2330
2220  REM  IF JOB NUMBER IS BLANK, DO NOT KEEP JOB INFORMATION
2230  IF JNO$ = "" THEN 2280
2240  REM  WRITE MODIFIED DATA TO SCRATCH FILE
2250  PRINT DOS$;"WRITE SCRATCH"
2260  PRINT JNO$: PRINT DESC$: PRINT DA$: PRINT AMOUNT: PRINT ST$
2270  REM  WRITE REMAINING DATA TO SCRATCH FILE
2280  PRINT DOS$;"READ ";NAM$
2290  INPUT JNO$: INPUT DESC$: INPUT DA$: INPUT AMOUNT: INPUT ST$
```

```
2300  PRINT DOS$;"WRITE SCRATCH"
2310  PRINT JNO$: PRINT DESC$: PRINT DA$: PRINT AMOUNT: PRINT ST$
2320  GOTO 2280
2330  POKE 216,0: PRINT DOS$;"CLOSE SCRATCH"
2340  PRINT DOS$;"CLOSE ";NAM$
2350  REM  ERASE ORGINAL FILE
2360  PRINT DOS$;"DELETE ";NAM$
2370  REM  RENAME SCRATCH FILE TO NAME OF ORIGINAL FILE
2380  PRINT DOS$;"RENAME SCRATCH,";NAM$
2390  GOTO 1130
2400  REM  ************************************************************
2410  REM  SUBROUTINE: OPEN FILE FOR INPUT
2420  ONERR  GOTO 2450
2430  PRINT DOS$;"APPEND ";NAM$
2440  GOTO 2480
2450  PRINT : PRINT "FILE DOES NOT EXIST"
2460  POKE 216,0
2470  GOTO 1130
2480  PRINT DOS$;"CLOSE ";NAM$
2490  PRINT DOS$;"OPEN ";NAM$: RETURN
```

Figure 13.3—Sequential File Management Program

As requested, we provided a modification option, called MODIFY. The job number is used as the key to finding the appropriate data in the file. We assumed that the job numbers are unique. The only way data in a sequential file can be modified is by rewriting the entire file. Therefore, this option uses two files, the existing data file and a temporary ''scratch'' file named "SCRATCH". Data are read from the existing data file and written to the temporary file until a match is made on the job number that was specified to be changed. At this point, the user is asked (prompted) for all of the information associated with this job. Next, this information is written to the temporary file followed by the remainder of the data that were in our original file. When the end of file is encountered, the original file is DELETEd (erased). The temporary file is then renamed with the name we had assigned to our original file. In this way we satisfy the requirement of rewriting the entire file whenever the contents of the sequential file are changed. At this point, control is returned to the menu.

Jobs can be removed from the data file by simply pressing the return key when the program prompts for the job number. This results in a job number having all blanks. The MODIFY option checks for a blank job number, and whenever one is found, the data associated with this job are not written to the new file.

One other subroutine is used in the program. This subroutine accesses the specified file using the append mode and checks to see if the file exists. Options 3 and 4 (APPEND and MODIFY) check the designated file using this subroutine. If the file does not exist, we know that we should not proceed with these options. Therefore, when a nonexisting file is detected, execution of these options is terminated and control is returned to the menu.

You might note that at the end of each option all open files are closed. This causes the disk operating system to write all data in the buffer to the disk file. As a result, if power to your computer is interrupted during program execution, the only data you will lose are those in a buffer associated with the option you are currently executing.

13.3 Random Access Files

The second type of file supported by Apple DOS is random access. Direct access is another term for this type of file. Random access files permit you to access data anywhere in a file without first reading all the preceding information in the file.

Writing to a Random Access File

Although random access files offer more flexibility, they require more programming steps to use. The following steps are required to write data to a random file:

1. Specify the file name.
2. Specify the record length.
3. Specify the write mode.
4. Specify the record number.
5. Write data to the file.
6. Close file.

The first two of the above steps are specified using an OPEN statement. For example, consider the following:

```
10 D$=CHR$(4): REM CTRL-D
20 PRINT D$; "OPEN PERSONNEL,L30,D2"
```

Statement 20 creates the file PERSONNEL on the diskette in drive 2, if it does not already exist. A record length of 30 characters (bytes) is also specified.

A RECORD is a group of logically related information. Each random access file is composed of a set of records. These records are numbered, starting with record number 0, then record number 1, and so forth. For instance, record 0 (in the PERSONNEL file) may contain information about some individual. Similarly, record 1 may contain information about another individual. So we can say that each record contains logically related information; the information contained in one record is related to one person. Thus, the information about one individual satisfies the definition of a record. We should note that the term "logically related information" can have a very general meaning, because how information is related depends on the particular application.

The next step in writing information to the PERSONNEL file requires that we specify the write mode and the location in the file (record number) where the record will be written. Each time a write occurs one record is written to the random access file. Consequently, the number of bytes specified in the length parameter (L) of the OPEN statement denotes the smallest unit of information that can be written to a file at one time. For example,

```
30 PRINT D$; "WRITE PERSONNEL,R1"
```

says that any following PRINT statement will write information in record 1 (R1 denotes the record number) of the PERSONNEL file. When a random access file is opened, the record pointer is set to the first record in the file.

Comparing statement 20 to the OPEN statement used for sequential files, you will note a difference, the record length specification. Here, a record length of 30 characters (bytes) was specified. You cannot specify the record length for sequential files. A method for calculating the record size for a random access file will be illustrated with an example. Like the sequential access method, the random access method requires read and write buffers; similarly they improve the efficiency of the disk input/output by reducing the number of disk accesses.

Only one more step remains in the process of storing information in a random access file, that of actually writing information to the file. This is accomplished using PRINT statements, such as:

```
40 PRINT X$: PRINT Y$
```

Statement 40 writes the strings X$ and Y$ to the record specified in the preceeding WRITE statement, which was record 1 in the example. Note that fields in a record are separated by carriage returns; the same way that fields are separated in a sequential file.

EXAMPLE 4. Write a program which will create a file for experimental data using a random access file. This file should provide space for the following types of data: parameter description, date, and parameter value.

SOLUTION. Before we write the program, we must determine how long to make each record. This can be done by determining the largest number of characters that would ever occur in each field. Assume that the following field sizes are required:

Field Description	Maximum Number of Characters
Parameter description	20
Date	8
Parameter value	5
Total	33

Thus, the description of a parameter will never take more than 20 characters; entering the date as MM/DD/YY requires 8 characters: and the largest parameter value that will occur is 99.99 (or any other combination of digits and a decimal point that does not exceed 5 characters). Remembering that each field is separated by a carriage return character, we must add one character to each field size. Consequently, we need to specify a record length of 38 bytes. However, if we ever try to store more than the specified length of characters in a record, we will lose some data. As a result, to provide some margin of error, we will specify a record length of 40 bytes. We can now write our program.

```
100 D$=CHR$(4): REM CTRL-D
110 PRINT D$; "OPEN EXPDATA,L40"
120 RECNO=0
130 INPUT "ENTER PARAMETER ?";NAM$
140 IF NAM$="END" THEN GOTO 240
150 INPUT "ENTER DATE ?";DA$
160 INPUT "PARAMETER VALUE ?";VLUE
170 REM WRITE DATA TO FILE
180 PRINT D$; "WRITE EXPDATA,R";RECNO
190 PRINT NAM$: PRINT DA$: PRINT VLUE
200 REM INCREASE RECORD NUMBER
210 RECNO=RECNO+1
220 PRINT D$
230 GOTO 130
240 PRINT D$; "CLOSE"
250 END
```

We specified a file name of EXPDATA and a record length of 40 characters. The variable RECNO was used to denote the record number in the file where the data are to be written.

If END is typed when the program prompts for the parameter description, the program terminates execution. Lines 180 and 190 place the data in the appropriate record.

Reading from a Random Access File

The steps required to read data from a random file are very similar to those for writing data. These steps are:

1. Specify the file name.
2. Specify the record length.
3. Specify the read mode.
4. Specify the record number.
5. Read data from file.
6. Close the file.

The first two steps are usually performed with the same statement (OPEN) that was used to write data to the random file. Consequently, if input and output to the same random file are performed in the same program, only one OPEN statement is required. One good rule to follow is always to read data from a file using the same format used to write it; for instance, do not change the record length. You will, therefore, want to use the same open statement.

To move the desired record from a random access disk file into memory, we use READ and INPUT statements:

```
30 PRINT D$; "READ EXPDATA,R1"
40 INPUT NAM$: INPUT DAT$: INPUT VLUE
```

Statement 30 specifies that subsequent INPUT statements are to read data from record 1 of the file named EXPDATA. Then, the next statement reads the information contained in this record.

EXAMPLE 5. Write a program which will access any desired record in the experimental data file created in Example 4 and print the contents of this record on your video display.

SOLUTION.

```
100 D$= CHR$(4): REM CTRL-D
110 PRINT D$; "OPEN EXPDATA,L40"
120 INPUT "ENTER NUMBER OF RECORD TO BE RETRIEVED ?";RECNO
130 IF RECNO<0 THEN GOTO 220
135 REM READ DATA FROM DISK
140 PRINT D$; "READ EXPDATA,R";RECNO
150 INPUT NAM$: INPUT DA$: INPUT VLUE
155 REM PRINT INFORMATION ON SCREEN
160 PRINT: PRINT "RECORD NO. =";RECNO
170 PRINT "PARAMETER : ";NAM$
180 PRINT "DATE : ";DA$
190 PRINT "VALUE :";VLUE
195 REM CANCEL THE DOS READ COMMAND
200 PRINT D$
210 GOTO 120
220 PRINT D$; "CLOSE"
230 END
```

Note that the records can be retrieved in any order. When we input a record number having a value less than 0, the program terminates. Also, note that line 200 cancels the DOS read command so that input can be obtained from the keyboard.

EXAMPLE 6. Combine the programs from Examples 4 and 5 such that after all the experimental data has been input, the user can print out the contents of any record.

SOLUTION.

```
]LIST

100 D$ =  CHR$ (4): REM  CRTL-D
110  PRINT D$;"OPEN EXPDATA,L40"
120 RECNO = 0
130  INPUT "ENTER PARAMETER ?";NAM$
140  IF NAM$ = "END" THEN  GOTO 240
150  INPUT "ENTER DATE ?";DA$
160  INPUT "PARAMETER VALUE ?";VLUE
170  REM  WRITE DATA TO FILE
180  PRINT D$;"WRITE EXPDATA,R";RECNO
190  PRINT NAM$: PRINT DA$: PRINT VLUE
210 RECNO = RECNO + 1
220  PRINT D$
230  GOTO 130
240  INPUT "ENTER NUMBER OF RECORD TO BE RETRIEVED ?";RECNO
250  IF RECNO < 0 THEN  GOTO 340
255  REM  READ DATA FROM DISK
260  PRINT D$;"READ EXPDATA,R";RECNO
270  INPUT NAM$: INPUT DA$: INPUT VLUE
275  REM  PRINT INFORMATION ON SCREEN
280  PRINT : PRINT "RECORD NO. =";RECNO
290  PRINT "PARAMETER : ";NAM$
300  PRINT "DATE : "DA$
310  PRINT "VALUE :";VLUE
315  REM  CANCEL THE DOS READ COMMAND
320  PRINT D$
330  GOTO 240
340  PRINT D$;"CLOSE"
350  END
```

Note that only one OPEN statement is required. Also, only one CLOSE statement was used.

Random File Management Program

It would require several thousand BASIC program statements to write a general random file access management program. Therefore, we will use an example that illustrates many of the concepts that are considered. The example will be the same one used to illustrate sequential access file management.

Before the details of the program are presented, we need to discuss how to locate a specific record. In Example 3, the Specialty Engineering Co. problem, we developed a program to store job related information. Assume that we have stored this information in a random access file. To locate a record with a particular job number, we will have to read one record at a time until we find the right job number. This is not an improvement over the sequential access method.

There are techniques which will improve this search process. One is the keyed access method. With this method we build a directory (table) containing job numbers and associated record numbers. The record number specifies where the job information for a

given job number is stored in the direct access file (See Figures 13.4(a) and 13.4(b)). When a job is to be stored in the file, the directory is searched for an unused record. The job information is then written to this location, and the directory is updated to show which job information is stored at this location. Figure 13.4(a) and 13.4(b) depict this process. If we want to locate a specific record (such as the record containing information for job B106) we can search the directory and see which record (record two for job B106) contains the desired information. This record can then be directly accessed.

One other problem remains. What do we do with the directory? It can be stored in a disk file or it can be reconstructed each time we use the program. Often it is stored at the beginning of the random access file for which it serves as a directory.

EXAMPLE 7. Reconsider Example 3 such that the data is stored in a random access file. The same types of data will be stored. The key to each record is the job number.

SOLUTION. Given the same information as in Example 3, we need to decide what record length should be used. Assume that the following field sizes will be used:

Field	Maximum Number of Characters/Field	
Job Number	4	
Job Description	12	
Due Date	8	(MM/DD/YY)
Bid Amount	9	
Status	2	
Total	35	

Adding 5 more characters to this total for the carriage returns that are used to delimit each field gives a value of 40 characters (bytes) per record. However, in order to provide some margin for error and to increase the generality of the program, a record size of 60 bytes will be specified.

We will slightly modify the procedure described above for storing the data record number in the directory. Looking at Figure 13.4, we can see that the record number corresponds to the position of a job number in the column denoted KEY. For instance, job number B106 was in position two in this column, and the associated data record used for this job is record number two. Therefore, the column used for record numbers is redundant; the position in the column KEY can be used to specify the record number.

In the data file in Figure 13.5, the first record is a directory record and the following records are used for data.

First, the directory is initialized so that all the fields contain "0000." This indicates that the associated data record is unused. As job information is entered, the appropriate job number is placed in the directory.

In Figure 13.5, the first field in the directory contains the job number A100. Since this is located in field number one of the directory, the associated data is found in the first data record, record number two in the file. The specific file record is found by adding:

Number of directory records	+	Field number in directory	=	File record number
1	+	1	=	2.

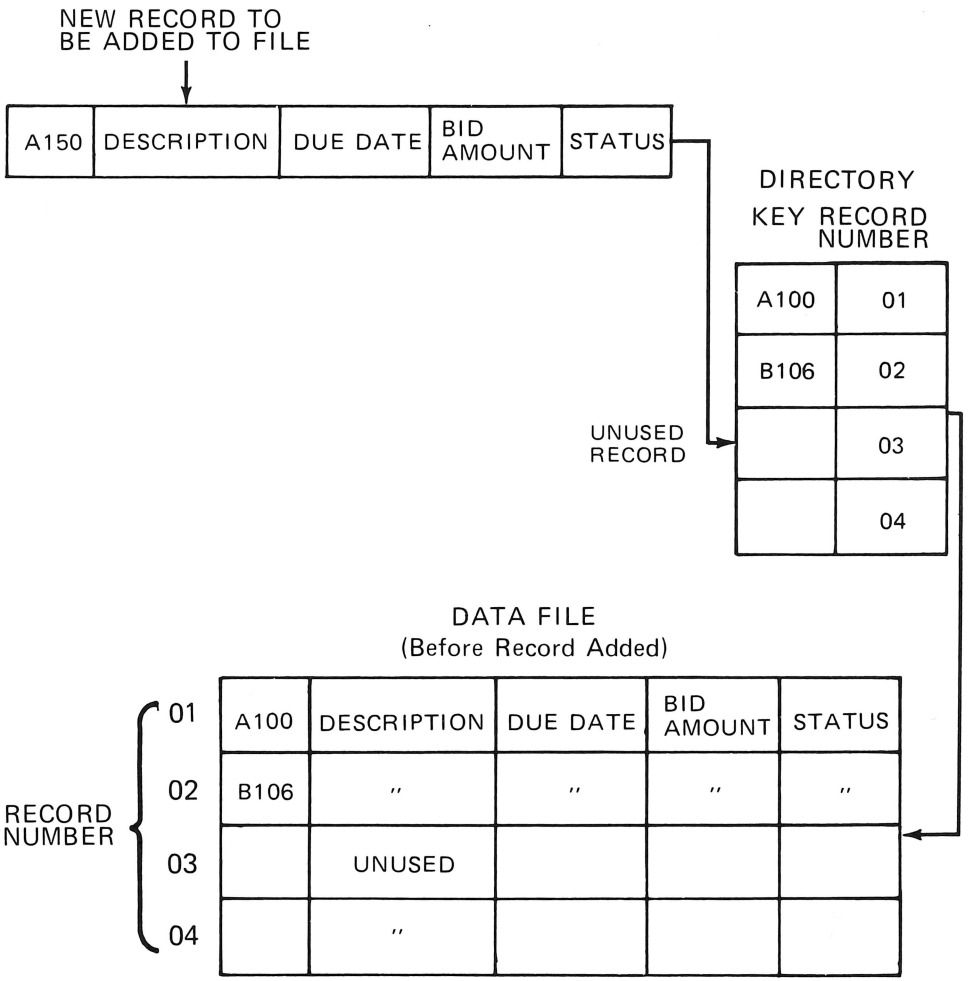


Figure 13.4(a)—Keyed Access Method Illustrated

In a similar manner we can calculate the file record number for job number A150:

$1 + 3 = 4$

Maintaining the directory in this manner saves space in the file.

For the Specialty Engineering Co. example, we will use a directory to maintain the location of each job related record as well as the location of each unused record. The

DIRECTORY

KEY	RECORD NUMBER
A100	01
B106	02
A150	03
	04

DATA FILE
(After Record Added)

RECORD NUMBER	A100	DESCRIPTION	DUE DATE	BID AMOUNT	STATUS
01	A100	DESCRIPTION	DUE DATE	BID AMOUNT	STATUS
02	B106	"	"	"	"
03	A150	"	"	"	"
04		UNUSED			

Figure 13.4(b)—Keyed Access Method Illustrated

directory will be stored at the beginning of the random access file, and the field number in the directory will designate the associated data record.

Now that a file index method has been defined, we need to determine how many directory records will be required, assuming that as many as 100 jobs may be active at one time. We specified that the size of the job number is 4 characters. Consequently, each directory record can contain 12 fields:

60 characters/record ÷ 5 characters/fields = 12 fields/record

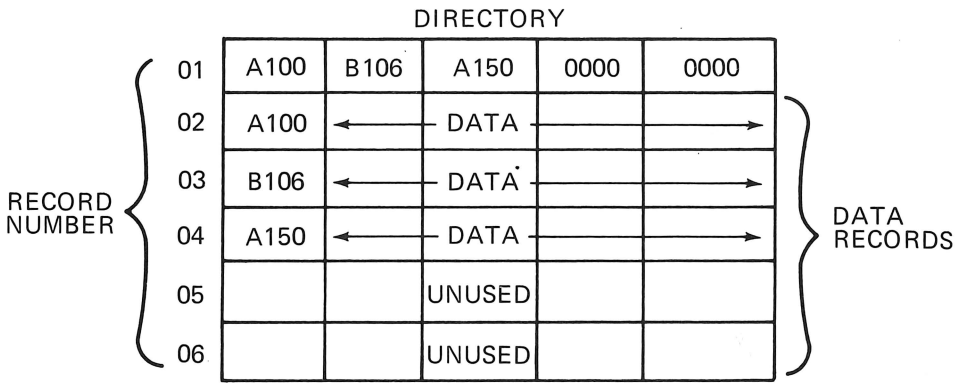


Figure 13.5—Example Directory and Associated Data Records

Note that 1 character was added to the field size to allow for the carriage return that delimits each field. We will use 10 directory records; therefore, we can maintain indices for 120 jobs. This number provides some margin for growth.

We will use a menu, like the one used in Example 3. The menu will consist of the following options:

1—Initialize Directory
2—New Job
3—Modify Job Data
4—Delete a Job
5—List Data
6—List Directory Contents
7—Exit Program

In the random access file management program, each of these options is a subroutine. The program appears in Figure 13.6. Each of these subroutines will be explained.

Option 1, INITIALIZE DIRECTORY, should only be used when you want to create a new job inventory file. This subroutine checks for a unique file name because the initialization procedure will destroy the contents of an existing file. If a file already exists, the user is given an opportunity to abort this option so that the contents of the existing file will be destroyed. The directory is initialized by writing '0000' in each field of each directory record. This value denotes that the associated data record is unused. After the directory is initialized, control is returned to the menu.

The second option, NEW JOB, is used to add new jobs to the file. This option prompts you for the data for a job and then locates an unused record in the data file. The job information is written into this record and the appropriate directory field is updated to denote the job data stored in the data record.

Job information could change (job status could change from working to completed, for example) or the original information could have been entered incorrectly. Option 3, MODIFY JOB DATA, may be used to change the data in the disk file. This option prompts you for the job number associated with the data to be changed. Using this

```
1000  REM      PROGRAM RANMAN
1010  REM      EXAMPLE RANDOM FILE DATA MANAGEMENT PROGRAM
1020  REM      MAINTAINS A JOB INVENTORY USING THE RANDOM ACCESS METHOD.
1030  REM      THE FOLLOWING INFORMATION IS STORED FOR EACH JOB:
1040  REM           1-JOB NUMBER
1050  REM           2-JOB DESCRIPTION
1060  REM           3-DUE DATE
1070  REM           4-BID DOLLAR AMOUNT
1080  REM           5-JOB STATUS
1090  REM       VARIABLE NAME      DESCRIPTION
1100  REM         IN PROGRAM      OF VARIABLE
1110  REM           JNO$             JOB NUMBER
1120  REM           DESC$            JOB DESCRIPTION
1130  REM           DA$              DUE DATE
1140  REM           AMOUNT           BID $ AMOUNT
1150  REM           SS$              JOB STATUS
1160  REM           DR$(I)           DIRECTORY FIELD I
1170  REM
1180 DOS$ =  CHR$ (4): REM  CTRL-D
1190 NF% = 12: REM     NO.FIELDS IN DIR.REC.
1200 NODIR = 10: REM      NO. OF DIRECTORY RECORDS
1210  DIM DR$(NF%)
1220  REM      HEADINGS FOR CRT OUTPUT
1230 H1$ = " NO.  DESCRIPTION    DATE     AMOUNT    ST"
1240 H2$ = "-----------------------------------------"
1250  PRINT : INPUT "INPUT DISK:FILE NAME TO BE USED FOR JOB INVENTORY ? ";NAM$
1260  PRINT : PRINT  TAB( 5);"MENU :": PRINT
1270  PRINT  TAB( 5)"1 - INITIALIZE DIRECTORY"
1280  PRINT  TAB( 5)"2 - NEW JOB"
1290  PRINT  TAB( 5)"3 - MODIFY JOB DATA"
1300  PRINT  TAB( 5)"4 - DELETE A JOB"
1310  PRINT  TAB( 5)"5 - LIST DATA"
1320  PRINT  TAB( 5)"6 - LIST DIRECTORY CONTENTS"
1330  PRINT  TAB( 5)"7 - EXIT PROGRAM"
1340  PRINT : INPUT "OPTION ? ";OPT
1350  IF (OPT < 1) OR (OPT > 7) GOTO 1430
1360  IF OPT = 1 THEN  GOTO 1450: REM   INITIALIZE DIRECTORY
1370  IF OPT = 2 THEN  GOTO 1750: REM   NEW JOB
1380  IF OPT = 3 THEN  GOTO 1900: REM   MODIFY JOB DATA
1390  IF OPT = 4 THEN  GOTO 2090: REM   DELETE A JOB
1400  IF OPT = 5 THEN  GOTO 2280: REM   LIST DATA
1410  IF OPT = 6 THEN  GOTO 2670: REM   LIST DIRECTORY CONTENTS
1420  IF OPT = 7 THEN  PRINT : PRINT  TAB( 9);"*** END OF PROGRAM ***": END
1430  PRINT : PRINT "INVALID OPTION, TRY AGAIN!": GOTO 1260
1440  REM     ********************************************************
1450  REM       INITIALIZE DIRECTORY
1460  REM     DETERMINE IF FILE NAME IS UNIQUE
1470  ONERR  GOTO 1560
1480  PRINT DOS$;"APPEND";NAM$
1490  PRINT DOS$;"CLOSE";NAM$
1500  POKE 216,0
1510  PRINT : PRINT "FILE ALREADY EXISTS WITH THIS NAME: ";NAM$
1520  PRINT : PRINT "INITIALIZING DIRECTORY WILL DESTROY OLD FILE ! ";
1530  INPUT "ABORT (Y/N) ? ";AN$
1540  IF AN$ <  > "N" THEN  GOTO 1260
1550  GOTO 1640
1560 Y =  PEEK (222): IF Y = 6 THEN  GOTO 1630
1570  REM     INVALID ERROR
1580  RESUME 1590
1590  POKE 216,0
1600  PRINT "ERROR OCCURRED WHILE ATTEMPTING TO OPEN FILE"
1610  GOTO 1260
1620  REM     VALID ERROR
1630  POKE 216,0
1640  PRINT DOS$;"OPEN";NAM$;",L60"
1650  FOR J = 1 TO NODIR
```

```
1660  PRINT DOS$;"WRITE";NAM$;",R";J
1670  FOR I = 1 TO NF%
1680 DR$(I) = "0000"
1690  PRINT DR$(I)
1700  NEXT I
1710  NEXT J
1720  PRINT DOS$;"CLOSE";NAM$
1730  GOTO 1260
1740  REM     ********************************************************
1750  REM     ADD A JOB TO FILE
1760  PRINT DOS$;"OPEN";NAM$;",L60": REM    OPEN FILE
1770  GOSUB 3100: REM     ASK FOR NEW DATA
1780 MH$ = "0000": REM     MATCH THIS VALUE IN DIR.
1790  GOSUB 3370: REM     OBTAIN UNUSED RECORD NO.
1800  IF IL <  > 0 THEN  PRINT "FILE FULL": GOTO 1260
1810  GOSUB 3030: REM      PLACE DATA IN RANDOM FILE
1820 DR$(IREC) = JNO$: REM     UPDATE DIRECTORY
1830  GOSUB 2890: REM      WRITE DIRECTORY TO DISK
1840  IF IL <  > 0 THEN  GOTO 1260: REM   IL<>0-INVALID RECORD NO.
1850  PRINT : INPUT "ADD ANOTHER JOB (Y/N) ? ";Y$
1860  IF Y$ = "Y" THEN 1770
1870  PRINT DOS$;"CLOSE"
1880  GOTO 1260
1890  REM     ********************************************************
1900  REM      MODIFY JOB INFORMATION
1910  PRINT DOS$;"OPEN";NAM$;",L60"
1920  PRINT : INPUT "ENTER JOB TO BE MODIFIED ? ";JNO$
1930 MH$ = JNO$: REM    MATCH THIS VALUE IN DIR.
1940  GOSUB 3370: REM    OBTAIN JOB RECORD NO.
1950  IF IL <  > 0 THEN  GOTO 1260: REM   IL<>0-NO MATCH
1960  GOSUB 3240: REM    READ DESIGNATED RECORD
1970  GOSUB 3310: REM    PRINT RECORD CONTENTS
1980  PRINT "ENTER MODIFIED JOB INFORMATION"
1990  GOSUB 3100: REM    ASK FOR INFORMATION
2000  GOSUB 3030: REM      PLACE DATA IN RANDOM FILE
2010 DR$(IREC) = JNO$: REM    UPDATE DIRECTORY
2020  GOSUB 2890: REM    WRITE DIRECTORY TO DISK
2030  IF IL <  > 0 THEN  GOTO 1260: REM    IL<>0-INVALID RECORD NO.
2040  PRINT : INPUT "ANY MORE JOBS TO BE MODIFIED (Y/N) ? ";Y$
2050  IF Y$ = "Y" GOTO 1920
2060  PRINT DOS$;"CLOSE"
2070  GOTO 1260
2080  REM     ********************************************************
2090  REM          DELETE A JOB
2100  PRINT DOS$;"OPEN";NAM$;",L60"
2110  PRINT : INPUT "JOB TO BE DELETED ? ";JNO$
2120 MH$ = JNO$: REM    MATCH THIS VALUE IN DIR.
2130  GOSUB 3370: REM    OBTAIN JOB RECORD NO.
2140  IF IL <  > 0 THEN  GOTO 1260: REM   IL<>0-NO MATCH
2150  GOSUB 3240: REM     READ DESIGNATED RECORD
2160  PRINT : PRINT "INFORMATION OF JOB TO BE DELETED"
2170  GOSUB 3310: REM      PRINT RECORD CONTENTS
2180  PRINT : INPUT "DELETE THIS RECORD (Y/N) ? ";Y$
2190  IF Y$ <  > "Y" THEN 2230
2200 DR$(IREC) = "0000": REM    UPDATE DIRECTORY
2210  GOSUB 2890: REM    WRITE DIRECTORY TO DISK
2220  IF IL <  > 0 THEN  RETURN
2230  PRINT : INPUT "DELETE ANOTHER JOB (Y/N) ? ";Y$
2240  IF Y$ = "Y" THEN 2110
2250  PRINT DOS$;"CLOSE"
2260  GOTO 1260
2270  REM     ********************************************************
2280  REM         LIST DATA
2290  PRINT DOS$;"OPEN";NAM$;",L60"
2300  PRINT : INPUT "ENTER JOB TO BE LISTED (ALL) ? ";JNO$
```

```
2310  IF JNO$ = "ALL" THEN 2410: REM   LIST ALL JOBS
2320 MH$ = JNO$: REM    MATCH THIS VALUE IN DIR.
2330  GOSUB 3370: REM    OBTAIN JOB RECORD NO.
2340  IF IL <  > 0 THEN  PRINT DOS$;"OPEN";NAM$;",L60": GOTO 2370: REM   NO MATCH
2350  GOSUB 3240: REM    OBTAIN DATA
2360  GOSUB 3310: REM    PRINT RECORD CONTENTS
2370  PRINT : INPUT "LIST ANOTHER JOB (Y/N) ? ";Y$
2380  IF Y$ = "Y" THEN 2300
2390  GOTO 2640
2400  REM      LIST ALL JOBS-STOP EVERY 20 JOBS TO READ DATA
2410 LINE = 1
2420 IC = 1
2430  PRINT : PRINT H1$: PRINT H2$
2440  FOR DC = 1 TO NODIR
2450  PRINT DOS$;"READ";NAM$;",R";DC
2460  REM     READ DIRECTORY RECORD
2470  FOR I = 1 TO NF%
2480  INPUT DR$(I)
2490  NEXT I
2500  PRINT DOS$
2510  FOR IREC = 1 TO NF%
2520  IF DR$(IREC) = "0000" THEN 2620: REM    RECORD UNUSED?
2530  REM      CALCULATE FILE RECORD NO.
2540 FF = NF% * (DC - 1) + IREC + NODIR
2550 LINE = LINE + 1
2560  IF LINE <  > (IC * 20) THEN 2600
2570 IC = IC + 1
2580  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
2590  PRINT : PRINT H1$: PRINT H2$
2600  GOSUB 3240: REM    OBTAIN DATA RECORD
2610  PRINT JNO$; TAB( 6);DESC$; TAB( 19);DA$; TAB( 28);AMOUNT; TAB( 38);SS$
2620  NEXT IREC
2630  NEXT DC
2640  PRINT DOS$;"CLOSE"
2650  GOTO 1260
2660  REM     ********************************************************
2670  REM           LIST DIRECTORY CONTENTS
2680  PRINT DOS$;"OPEN";NAM$;",L60"
2690  PRINT : PRINT  TAB( 11);"DIRECTORY CONTENTS"
2700  PRINT  TAB( 10);"==================="
2710  FOR DC = 1 TO NODIR
2720  PRINT "DIRECTORY RECORD NO. ";DC
2730  PRINT : PRINT "FIELD NO.","FIELD VALUE"
2740  PRINT DOS$;"READ";NAM$;",R";DC
2750  FOR C = 1 TO NF%
2760  INPUT DR$(C)
2770  NEXT C
2780  FOR IREC = 1 TO NF%
2790 FF = NF% * (DC - 1) + IREC
2800  PRINT DOS$
2810  PRINT FF,DR$(IREC)
2820  NEXT IREC
2830  REM      STOP AFTER EVERY DIRECTORY RECORD TO READ DATA
2840  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";Y$: PRINT
2850  NEXT DC
2860  PRINT DOS$;"CLOSE"
2870  GOTO 1260
2880  REM   ***********************************************
2890  REM     SUBROUTINE: WRITE DIRECTORY RECORD TO DISK
2900 IL = 0: REM    INDICATES NO ERROR
2910  IF (DC > 0) AND (DC < (NODIR + 1)) THEN 2960
2920  PRINT "INVALID DIRECTORY RECORD"
2930 IL = 1
2940  PRINT DOS$;"CLOSE"
2950  RETURN
```

```
2960  PRINT DOS$;"WRITE";NAM$;",R";DC
2970  FOR I = 1 TO NF%
2980  PRINT DR$(I)
2990  NEXT I
3000  PRINT DOS$
3010  RETURN
3020  REM     ************************************************************
3030  REM     SUBROUTINE: WRITE DATA TO RANDOM FILE
3040  PRINT DOS$;"WRITE";NAM$;",R";FF
3050  PRINT JNO$: PRINT DESC$: PRINT DA$
3060  PRINT AMOUNT: PRINT SS$
3070  PRINT DOS$
3080  RETURN
3090  REM     ************************************************************
3100  REM     SUBROUTINE: ASK FOR JOB INFORMATION
3110  PRINT : INPUT "ENTER JOB NUMBER: ";JNO$
3120  IF  LEN (JNO$) > 4 THEN  PRINT : PRINT "ONLY 4 DIGITS ALLOWED": GOTO 3110
3130  PRINT : INPUT "ENTER JOB DESCRIPTION ? ";DESC$
3140 DESC$ =  LEFT$ (DESC$,12)
3150  PRINT : INPUT "ENTER DUE DATE ? ";DA$
3160 DA$ =  LEFT$ (DA$,8)
3170  PRINT : INPUT "ENTER AMOUNT BID ?  $ ";AMOUNT
3180 A$ =  STR$ (AMOUNT)
3190  IF  LEN (A$) > 9 THEN  PRINT : PRINT "ONLY 9 DIGITS ALLOWED": GOTO 3170
3200  PRINT : INPUT "JOB STATUS ? ";SS$
3210  IF  LEN (SS$) > 2 THEN  PRINT : PRINT "ONLY 2 CHARACTERS ALLOWED": GOTO 3200
3220  RETURN
3230  REM     ************************************************************
3240  REM     SUBROUTINE: READ A DATA RECORD FROM RANDOM FILE
3250  PRINT DOS$;"READ";NAM$;",R";FF
3260  INPUT JNO$: INPUT DESC$: INPUT DA$
3270  INPUT AMOUNT: INPUT SS$
3280  PRINT DOS$
3290  RETURN
3300  REM     ************************************************************
3310  REM     SUBROUTINE: PRINT RECORD CONTENTS ON MONITOR
3320  PRINT : PRINT H1$: PRINT H2$
3330  PRINT JNO$; TAB( 6);DESC$; TAB( 19);DA$; TAB( 28);AMOUNT; TAB( 38);SS$
3340  PRINT
3350  RETURN
3360  REM     ************************************************************
3370  REM     SUBROUTINE: MATCH JOB NO. IN DIRECTORY
3380 IL = 0: REM   INDICATES NO ERROR
3390  FOR DC = 1 TO NODIR
3400  PRINT DOS$;"READ";NAM$;",R";DC
3410  FOR C = 1 TO NF%
3420  INPUT DR$(C)
3430  NEXT C
3440  PRINT DOS$
3450  FOR IREC = 1 TO NF%
3460  IF DR$(IREC) <  > MH$ THEN 3500
3470  REM      CALCULATE FILE RECORD NO.
3480 FF = NF% * (DC - 1) + IREC + NODIR
3490  GOTO 3550
3500  NEXT IREC
3510  NEXT DC
3520  PRINT "NO MATCH FOUND FOR JOB ID: ";MH$
3530  PRINT DOS$;"CLOSE"
3540 IL = 1: REM  INDICATES ERROR
3550  RETURN
```

Figure 13.6—Random File Management Program

number, the directory is searched to find where the job data record is stored. This search is performed by reading one directory record at a time and comparing each field in the directory with the specified job number. If no match is found in any of the directory records, the following message is displayed:

```
NO MATCH FOUND FOR JOB ID: XXXX
```

where XXXX is the job number. When the specified job number is found in the directory, the program prompts you for the data that will replace the existing job data.

Option 4, DELETE JOB, permits you to remove a job from the file. You are prompted for the job number. When the record containing this job information is located, the information is displayed. The program then asks if you want to delete this information. In this way you get a chance to review the data before they are deleted. If you reply with a "YES," the value '0000' is written into the directory field associated with this data record. Consequently, the status of this data record is now unused.

The fifth option, LIST DATA, will list the contents of the data records. The job information for a particular job number can be displayed. If you respond with 'ALL' when prompted for the job number, the job information for all jobs in the file will be listed. The contents of the directory can be listed using option 6, LIST DIRECTORY CONTENTS. This option displays all fields in the directory one record at a time.

The last option, EXIT PROGRAM, lets you terminate the program. You should note that as each option is completed, the open file is closed. This avoids the problem of determining if the data file is open each time a new option is specified. Using this procedure, we know that the file is not open. This also causes the contents of the buffer to be written to the data file.

The subroutine starting at line 2890 is used to write directory records to the data file, and the subroutine starting at line 3030 is used to write job information to the data file. Note that we have two types of records in the same data file, which is valid as long as neither violates the record length specification of 60 bytes.

A modular approach was used to develop this program: each option consists of one or more subroutines. This made the program easier to write and easier to debug. Also note that 120 jobs can now be accommodated (10 directory records $\times$ 12 job fields per record). This value can be changed by changing only one variable, NODIR, which specifies the number of directory records. It now has a value of 10.

In the last example, a keyed index method was used to maintain the location of the data records. Other methods, such as hashing, can be used. Space limitations do not permit an explanation of this technique. The interested reader might refer to Baase (1978). Data base management systems (DBMS) are also available. These systems are designed to handle a wide range of sequential and random file applications. Initially DBMS were available only for minicomputers and main frame computers. They are now becoming available for microcomputers. For a general description of DBMS, refer to Ross (1978).

13.4 Summary

Two methods for storing data in disk files were presented in this chapter. We showed how to create and use sequential and random access disk files. A keyed index method was discussed that could be used to quickly locate records in a random access file.

Although the disk input/output programs presented are not very general, they nonetheless will be useful guides as you develop programs of your own.

References

Baase, Sara, *Computer Algorithms: Introduction to Design and Analysis*, Addison-Wesley, Massachusetts, 1978.

Bradley, James, *File and Data Base Techniques*, Holt, Reinhart and Winston, New York, 1981.

Ross, Ronald G., *Data Base Systems Design, Implementation and Management*, AMACOM, New York, 1978.

Exercises

1. Give some examples of primary storage devices. Give a few examples of secondary storage devices.
2. What is a *sector* on a floppy disk? How many sectors are on one side of an Apple floppy diskette?
3. What is the primary function of the DOS diskette directory?
4. What is the sequential access method? What is the random access method?
5. Describe the input/output modes that can be used with the sequential access method.
6. For random access files, define the following terms:
   a. record
   b. buffer
   c. file
7. Data for several experimental tests along with the date of the tests are to be stored in a disk file. At this time, the following data are available:

Date	Test 1	Test 2	Test 3
2/15/83	100.4	96.7	105.2
2/18/83	151.0	159.3	165.8
3/1/83	125.9	117.4	110.6

   Write a program which will
   (a) create a sequential data file named TESTDATA on disk drive 2, and
   (b) store the above test data in this file.
8. Write a program which will add the following data to the file created in Exercise 7.

Date	Test 1	Test 2	Test 3
3/15/83	172.4	185.9	190.6
3/22/83	200.6	205.0	203.7

9. Write a program which will
   a. create a random access file named OUTCOMES on disk drive 2, and
   b. store the data from Exercise 7 in this file.
10. Write a program which will display on your printer the contents of the file created in Exercise 7.
11. Write a program which will display on your printer the contents of the file created in Exercise 9.

14

Data Structures

Small amounts of data can be stored on top of a desk; however, as the amount of information increases, a file cabinet will be required. Decisions must then be made regarding how information will be stored. If the data are well organized, a given item can be located in less time and less storage space is required.

Working with data within a computer is similar. When working with a small amount of data, we usually are not concerned with how the data are stored (structured). However, when substantial amounts of data are involved, we learn that the data structure can dramatically affect the program execution time, amount of storage required, ease of use, and ease of programming.

14.1 Arrays

Several techniques have been developed to aid in structuring data. Some of the simplest to implement will be discussed in this chapter. One such structure is the *array*. An array is a group of data elements referenced by the same name, A. Each element is referenced by specifying its associated subscript; thus A(3) specifies the third element in the array A. The array A(I) is called a one-dimensional array. Likewise, A(I,J) is a two-dimensional array. Using Applesoft, an array can have 255 dimensions; however, arrays having more than three dimensions are seldom used.

Use of an array can greatly simplify a programming task. When initializing the value of a series of variables, using an array reduces the number of statements required. For example, consider the problem of initializing 20 variables to the value of 100, using an array:

```
10 DIM A(20)
20 FOR I = 1 TO 20
30 A(I) = 100
40 NEXT I
```

Without an array, 20 assignment statements would have been required to perform this task:

```
A1 = 100
A2 = 100
```

.
.
.

A20 = 100

An array is the simplest of data structures, and it is the structure most often used. In fact, arrays are used in most of the programs in this book.

14.2 Stacks

The second data structure we will consider is called a *stack*. The operation of a stack can be visualized by considering a springloaded cafeteria tray holder. When trays are placed in a tray holder, the trays descend until only the top tray is visible. As a tray is removed from the top, the remaining trays move up so that the top tray is visible. This is also an example of last-in, first-out (LIFO) storage. A stack data structure works in a similar manner. All data additions to and deletions from the stack occur at one end. Adding data to the stack is often called "pushing" the stack; removing data is called "popping" the stack.

In a BASIC program, an array is often used to implement a stack. Data elements are stored in the array. Moving the data within an array each time an element is added or deleted (similar to the cafeteria tray holder) can be avoided by using a variable (sometimes called a stack pointer) to denote the "top of stack." Consider the example in Figure 14.1 where three data elements are added to an empty stack and then two elements are removed.

A stack data structure is used by the central processing unit of most computers to store information temporarily as instructions are executed. This structure is widely used because it requires very few program statements to implement. Within a program, two of the more common applications of a stack data structure are the maintenance of unused space in an array and unused records in a disk file. In a following section of this chapter, there is a program which uses a stack to maintain a list of unused space in an array. Figure 14.2 contains three subroutines which will assist you in using a stack data structure. Note the small number of BASIC statements required.

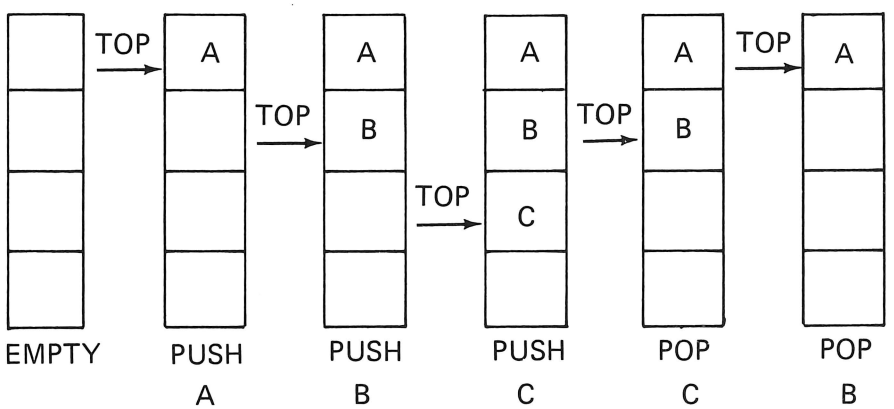


Figure 14.1—Stack Structure

```
24500  REM  STACK SUBROUTINES
24510  REM  THESE SUBROUTINES WILL INITIALIZE A STACK AND
24520  REM  PERFORM DATA ADDITIONS AND DELETIONS TO
24530  REM  ARRAY SK. TAP IS THE STACK POINTER. SIZ IS
24540  REM  THE MAXIMUM SIZE OF THE STACK.DATA TO BE
24550  REM  PLACED IN THE STACK RESIDES IN IDTA.
24560  REM  ************************************************
24570  REM  SUBROUTINE: STACK INITIALIZE
24580  DIM SK(SIZ%): REM  DEFINE ARRAY SIZE
24590 TAP% =  - 1: REM  DENOTE EMPTY STACK
24600  RETURN
24610  REM  ************************************************
24620  REM  SUBROUTINE: STACK PUSH
24630 IL = 0: REM  SET NO ERROR CONDITION
24640  IF (TAP% >  = SIZ%) THEN 24680: REM  STACK FULL?
24650 TAP% = TAP% + 1: REM  PUSH DATA
24660 SK(TAP%) = IDTA
24670  RETURN
24680  PRINT : PRINT "STACK OVERFLOW"
24690 IL = 1: RETURN : REM  SET ERROR CONDITION
24700  REM  ************************************************
24710  REM  SUBROUTINE: STACK POP
24720 IL = 0: REM  SET NO ERROR CONDITION
24730  IF (TAP% <  =  - 1) THEN 24770: REM  STACK EMPTY ?
24740 IDTA = SK(TAP%): REM  POP DATA
24750 TAP% = TAP% - 1
24760  RETURN
24770  PRINT :"STACK UNDERFLOW"
24780 IL = 1: RETURN : REM  SET ERROR CONDITION
```

Figure 14.2—Stack Subroutines

In these subroutines the stack is maintained in the array SK and the maximum size (SIZ) of this array is supplied by the main program. In the subroutine STACK INITIALIZATION, all the elements in the array SK are equated to 0 by the statement:

DIM SK(SIZ)

In BASIC the minimum value for an array subscript is (0); therefore, to show that the stack is empty, the stack pointer TOP is initially set equal to (−1). Subroutine STACK POP will remove a data element from the stack and place the value in IDAT. However, before an element is removed, the stack pointer is evaluated to see if the stack is empty. If it is empty, an error indicator, IL, is set equal to (1) and a return from subroutine occurs. Otherwise, IL remains equal to (0), indicating no error, and an element is removed.

Subroutine STACK PUSH will add data to the stack. Before this occurs, the stack pointer is checked to determine if adding one more data element will exceed the maximum size of the array. If so, IL is set equal to (1) and a return from subroutine occurs.

EXAMPLE 1. Write a program that will use the subroutines in Figure 14.2 to store the three data elements A = 10, B = 5, and C = 25 in a stack. At this point, your program should print out the contents of the stack; then elements C and B are to be removed, printing each element as it is removed. Next, print the final contents of the stack. Assume that all data stored in the stack will be integers.

SOLUTION. Your response to Example 1 might look like the program in Figure 14.3. There are many ways to write this program. The maximum size (SIZ) of the stack array (SK) was set as 6.

```
100  REM  STACK EXAMPLE PROGRAM
110 SIZ% = 6: REM                          SET ARRAY SIZE
120  GOSUB 24560: REM                       INITIALIZE STACK
130 A = 10
140 B = 5
150 C = 25
160 IDTA = A
170  GOSUB 24620: REM                       PUSH A ON STACK
180  IF IL <  > 0 THEN 410
190 IDTA = B
200  GOSUB 24620: REM                       PUSH B ON STACK
210  IF IL <  > 0 THEN 410
220 IDTA = C
230  GOSUB 24620: REM                       PUSH C ON STACK
240  IF IL <  > 0 THEN 410
250  FOR I = 0 TO TAP%
260  PRINT "STACK",I,SK(I): REM             PRINT STACK CONTENTS
270  NEXT I
280  GOSUB 24710: REM                       POP C FROM STACK
290  IF IL <  > 0 THEN 410
300 C = IDTA
310  PRINT : PRINT "C",,C
320  GOSUB 24710: REM                       POP B FROM STACK
330  IF IL <  > 0 THEN 410
340 B = IDTA
350  PRINT "B",,B
360  FOR I = 0 TO TAP%
370  PRINT : PRINT "STACK",I,SK(I): REM     PRINT STACK CONTENTS
380  NEXT I
390  STOP
400  PRINT : PRINT "ERROR"
410  END

]
```

Figure 14.3—Using Stack Subroutines

14.3 Queues

A data structure that processes data in a first-in, first-out (FIFO) order is called a queue. Additions to data are made at one end of the structure and all deletions are made at the other end. You might relate this structure to a waiting line of people (a queue in front of a bank teller). New arrivals join the *rear* of the line, and as people are serviced, they leave the *front* of the line. We can use an array to implement a queue structure. In addition, two pointers are needed; one to denote the front of the queue (F) and one to denote the rear (R). Figure 14.4 illustrates data elements being added and removed from a queue.

Note that the ends of the data structure move from left to right across the array. When the end of the array is reached, the next data element to be added to the structure will be placed in the first cell of the array, if that location is empty. Figure 14.5 illustrates this process.

Before writing the programs to implement a queue structure, we need to decide how to determine when the array holding the data is empty or full. Looking at Figure 14.4, you can see that an empty queue occurred when the rear pointer equaled the front pointer ($R = F$). If we adopt this convention, we could also say that the queue is full when $R = F - 1$, such as in Figure 14.5c. These conventions mean that only $(n - 1)$ data elements

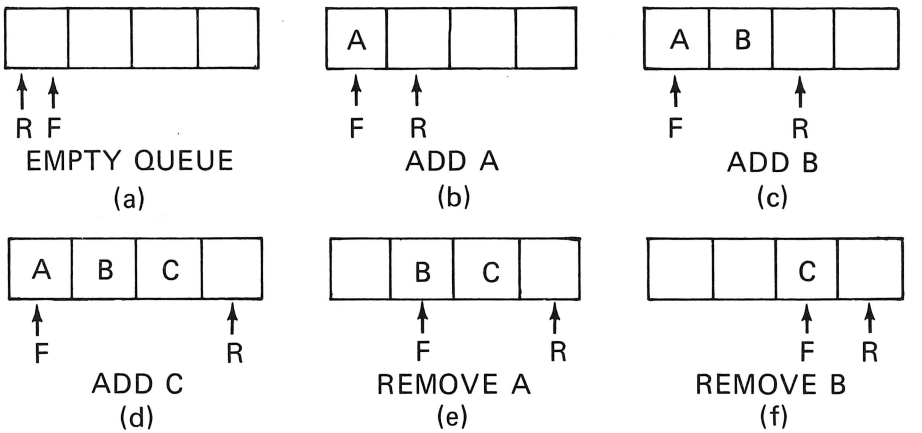


Figure 14.4—Queue Structure

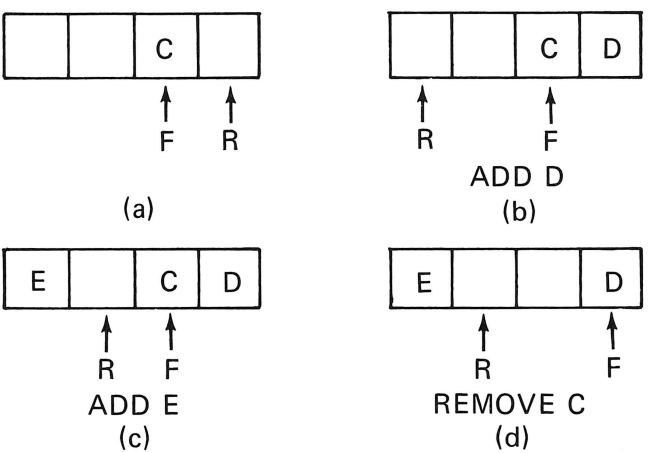


Figure 14.5—Additional Queue Structure Illustrations

can be stored in an array with n cells; however, maintaining the simplicity is worth losing one memory location. Figure 14.6 contains three subroutines that will assist you in using a queue data structure. Like the stack related routines, these programs will initialize a queue structure and perform additions and deletions to a queue.

EXAMPLE 2. Write a program that will use the subroutines in Figure 14.6 to store the three data elements A = 10, B = 5, and C = 25 in a queue data structure. At this point, print out the contents of the queue. Then remove elements A and B, displaying the respective values as each is removed. Next, print the contents of the queue.

```
24800  REM  QUEUE SUBROUTINES
24810  REM  THESE SUBROUTINES WILL INITIALIZE A QUEUE
24820  REM  AND PERFORM DATA ADDITIONS AND DELETIONS
24830  REM  TO A QUEUE. THE QUEUE IS MAINTAINED IN
24840  REM   THE ARRAY Q. FQ% IS THE FRONT OF QUEUE
24850  REM  POINTER, AND RQ% IS THE REAR OF QUEUE POINTER.
24860  REM  ********************************************************
24870  REM  SUBROUTINE: QUEUE INITIALIZE
24880  DIM Q(QSIZ%)
24890 FQ% = 0: REM  DENOTE EMPTY QUEUE
24900 FQ% = RQ%
24910  RETURN
24920  REM  ********************************************************
24930  REM  SUBROUTINE: QUEUE ADD
24940 IL = 0: REM  SET NO ERROR CONDITION
24950  IF RQ% = (FQ% - 1) THEN 25010: REM  IS QUEUE FULL?
24960  IF (FQ% = 0) AND (RQ% = QSIZ%) THEN 25010
24970 Q(RQ%) = IDTA: REM  ADD TO QUEUE
24980 RQ% = RQ% + 1
24990  IF RQ% > QSIZ% THEN RQ% = 0
25000  RETURN
25010  PRINT : PRINT "QUEUE FULL"
25020 IL = 1: RETURN : REM  SET ERROR CONDITION
25030  REM  ********************************************************
25040  REM  SUBROUTINE: QUEUE DELETE
25050 IL = 0: REM  SET NO ERROR CONDITION
25060  IF RQ% = FQ% THEN 25110: REM  IS QUEUE EMPTY ?
25070 IDTA = Q(FQ%)
25080 FQ% = FQ% + 1
25090  IF FQ% > QSIZ% THEN FQ% = 0
25100  RETURN
25110  PRINT : PRINT "QUEUE EMPTY"
25120 IL = 1: RETURN
```

Figure 14.6—Queue Subroutines

SOLUTION. Figure 14.7 contains a program that is one possible solution. The queue data array, Q, is dimensioned to six by setting QSIZ=6. In this program, data to be input to the queue are read into the variable IDTA using a subroutine residing at line 230 (called INPUT TO QUEUE). Each time a data element is input, this subroutine is used. Following the read statement in this subroutine, subroutine QUEUE ADD from Figure 14.6 is referenced. This latter subroutine places the data into the queue data array.

A subroutine has also been written to print the queue (called QUEUE PRINT) contents; this subroutine starts at line 350. A third subroutine (called TAKE FROM QUEUE) has been written for this example. It starts at line 300 and is referenced every time a data element is removed from the queue. This subroutine references subroutine QUEUE DELETE from Figure 14.6. A data element removed from a queue is placed in the variable IDTA.

14.4 Linked Lists

A *list* is an ordered collection of data elements. In a *linked list*, each element has an accompanying *pointer*, which specifies the next element in the list. As a result, although the data is not physically stored in a specified order, it can be logically processed in the proper order using the pointers.

```
100  REM  QUEUE EXAMPLE PROGRAM
110 QSIZ% = 6: REM                          SET ARRAY SIZE
120  GOSUB 24870: REM                      INITIALIZE QUEUE
130  GOSUB 230: REM                        PLACE DATA IN QUEUE
140  GOSUB 350: REM                        PRINT QUEUE CONTENTS
150  GOSUB 300: REM                        TAKE DATA FROM QUEUE
160  PRINT "A = ";IDTA
170  GOSUB 300
180  PRINT "B = ";IDTA: PRINT
190  GOSUB 350: REM                        PRINT QUEUE CONTENTS
200  DATA  10,5,25,-9999
210  END
220  REM  ******************************************************
230  REM  SUBROUTINE: INPUT TO QUEUE
240  READ IDTA
250  IF IDTA =  - 9999 THEN  RETURN
260  GOSUB 24930
270  IF IL <  > 0 THEN  PRINT "ERROR": END
280  GOTO 240
290  REM  ******************************************************
300  REM  SUBROUTINE: TAKE FROM QUEUE
310  GOSUB 25040
320  IF IL <  > 0 THEN  PRINT "ERROR": END
330  RETURN
340  REM  ******************************************************
350  REM  SUBROUTINE: QUEUE PRINT
360  PRINT "QUEUE CONTENTS-FIFO ORDER"
370  PRINT : PRINT "ARRAY LOCATION","VALUE"
380  IF FQ% = RQ% THEN 450
390 I = FQ%
400  PRINT I,Q(I)
410 I = I + 1
420  IF I > QSIZ% THEN I = 0
430  IF I = RQ% THEN  PRINT : RETURN
440  GOTO 400
450  PRINT "QUEUE EMPTY": PRINT : RETURN

]
```

Figure 14.7—Using Queue Subroutines

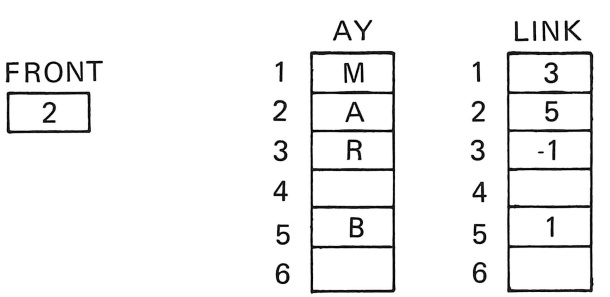


Figure 14.8—Linked List Structure

Figure 14.8 contains an example of a linked list ordered alphabetically. Using the pointers contained in the box labeled FRONT and the array LINK, the data elements in the array AY can be processed in alphabetical order.

FRONT denotes the location of the first data element in the list (located in cell 2 of the array AY). The number (5) in cell 2 of the array LINK (adjacent to position 2 of the array AY) points to the location of the next data element in alphabetical order. This process continues until the pointer value is a (−1), which indicates "end of list." In Figure 14.9, a new data element has been added to the list.

Most sets of data are dynamic in that over time, data are added and deleted. Maintaining data in a specific order, such as alphabetical, can involve moving the data elements as deletions and additions occur. However, if a linked list data structure is used, we need not move any of the data elements when the contents of the set of data are changed. Instead, the linked list pointers are modified to reflect any changes.

It is also advantageous to use a linked list when a limited amount of storage space is available for data lists containing varying numbers of elements. An example would be a block of storage space allocated to storing the names of employees in each department of some firm. Each employee in a particular department must be associated with the right department. However, the number of employees in a department may vary from small to large depending on the size of the department. Another consideration is the dynamic nature of the data; new employees must be added and terminated employees must be removed. If we reserve a block of space in an array for each department, each space must be large enough to handle the largest department. This will result in unused space for the small departments. If we use a linked list data structure for this type of problem, we can efficiently utilize our storage space while keeping our program execution times small.

A linked list data structure permits more than one list to be stored in a common block of storage space. Figure 14.10 illustrates an array that holds data for two linked lists. Note the FRONT pointer associated with each list in the array. A stack data structure has been utilized to provide a list of unused cells in the data array in this figure.

A stack is a convenient method of keeping track of unused space in a block of storage space (memory or disk). As an element is deleted from the block of storage, the unused space is added to the top of the stack. Another data element is added to the data array in the location specified by the number at the top of the stack. For example, if another data element were added to the data array in Figure 14.10, it would be placed in row 3, and the top of stack would then point to the number 8 in the stack.

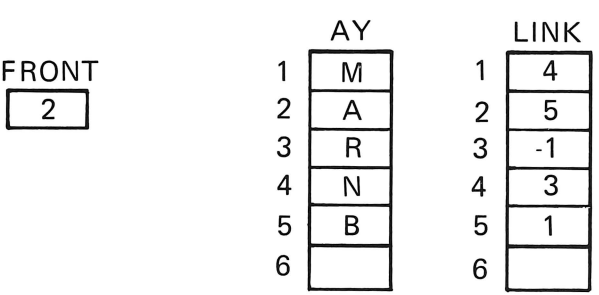


Figure 14.9—Additional Linked List Illustrations

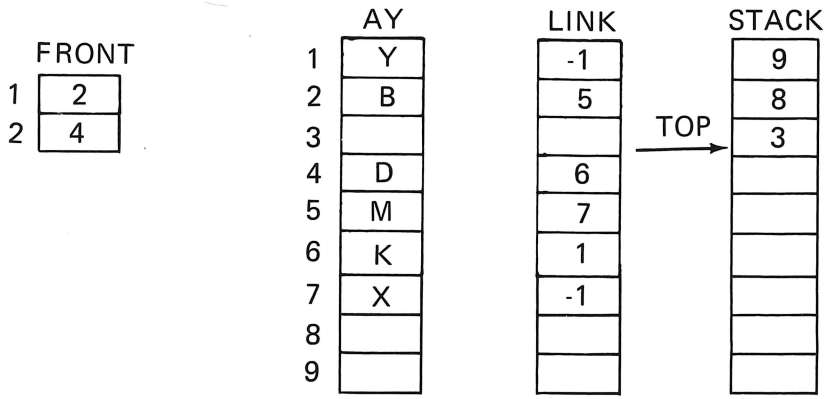


Figure 14.10—Multiple Linked Lists

Linked Queues

We have been discussing linked lists, saying that each list is an ordered collection of data elements. How the lists are ordered (LIFO, FIFO, alphabetical, etc.) depends upon the particular application. Also associated with each element in a linked list might be several other data fields. An example might be an alphabetical list of employees in a department. With each name, we may want to maintain the social security number, address, and age.

Figure 14.11 contains programs that can be used to implement linked lists when each list is a queue. The data elements are maintained in the array LQDTA; each element in an individual queue is linked using the array, LINK. The front and rear of each queue are maintained in the arrays FRNT and REAR.

Because more than one queue may be stored in the array LQDTA, the convention described in Section 14.4 cannot be used to denote an empty or full queue ($R = F$ and $R = F - 1$). Instead, the program in Figure 14.11 uses the stack routines in Figure 14.2 to maintain a list of available space in the array LQDTA. Initially, the stack will contain all cells in the data array LQDTA because the data array is empty. The queue initialization subroutine places all of the cell numbers available in LQDTA in the stack. No more data can be added to a queue if the stack indicates that no space is available in the array LQDTA. We will know that queue I is empty when the variable FRNT(I) $= -1$.

EXAMPLE 3. Write a program which will put the following data in a linked list of queues:

Queue Number	Data
1	50
1	10
2	75
3	113

```
25200  REM  SUBROUTINES FOR LINKED LISTS (QUEUES)
25210  REM  THESE ROUTINES WILL INITIALIZE, PERFORM ADDITIONS AND
25220  REM  DELETIONS TO LINKED LISTS OF QUEUES. THE DATA RESIDE IN
25230  REM  AN ARRAY CALLED LQDTA%; THESE DATA ARE LINKED INTO FIFO
25240  REM  QUEUES USING THE ARRAY LINK%. FRNT%(I) POINTS TO FRONT
25250  REM  OF QUEUE I; REAR% POINTS TO END OF QUEUE I. LOCATION IN
25260  REM  ARRAY LQDTA% OF DATA TO BE ADDED OR REMOVED IS INDICATED
25270  REM  BY PT%. A LIST OF AVAILABLE SPACE IN THE DATA ARRAY IS
25280  REM  MAINTAINED IN STACK ARRAY, SK.
25290  REM  ******************************************************
25300  REM  SUBROUTINE: INITIALIZE LINKED QUEUES AND STACK
25310  DIM LQDTA%(SIZ%),LINK%(SIZ%)
25320  DIM FRNT%(NQ%),REAR%(NQ%)
25330  REM  MUST SET FRONT AND REAR POINTERS TO -1 TO
25340  REM  INDICATE EMPTY QUEUES.
25350  FOR I = 1 TO NQ%
25360 FRNT%(I) =  - 1
25370 REAR%(I) =  - 1
25380  NEXT I
25390  REM  PLACE ALL AVAILABLE DATA LOCATIONS IN STACK.
25400  REM  WANT TO USE LOWER NUMBERED CELLS FIRST (LIFO)
25410  FOR I = SIZ% TO 0 STEP  - 1
25420 IDTA = I
25430  GOSUB 24620: REM  PLACE DATA ELEMENT IN STACK
25440  NEXT I
25450  RETURN
25460  REM  SUBROUTINE: ADD DATA TO A LINKED QUEUE
25470 IL = 0: REM  SET NO ERROR CONDITION
25480  GOSUB 24710: REM   OBTAIN AVAILABLE DATA CELL LOCATION
25490  IF IL <  > 0 THEN  PRINT "ERROR STACK ROUTINE": END
25500 PT% = IDTA: REM   LOCATION OBTAINED FROM STACK
25382  IF (PT% < 0) OR (PT% > SIZ%) THEN 25610: REM   DOES LOCATION EXIST ?
25520  REM  IF QUEUE NOT EMPTY, LINK TO REAR OF LIST
25530  IF REAR%(IQ) <  >  - 1 THEN LINK%(REAR%(IQ)) = PT%
25540 LINK%(PT%) =  - 1: REM  DENOTE END OF LIST
25550 REAR%(IQ) = PT%
25560  REM  IF QUEUE WAS EMPTY, MUST SET FRONT POINTERS
25570  IF FRNT%(IQ) =  - 1 THEN FRNT%(IQ) = PT%
25580 LQDTA%(PT%) = LDTA%
25590  RETURN
25600 IL = 1: REM  SET ERROR CONDITION
25610  PRINT : PRINT "INVALID LOCATION IN DATA ARRAY": RETURN
25620  REM  ******************************************************
25630  REM  SUBROUTINE: DELETE DATA FROM A LINKED QUEUE
25640 IL = 0: REM  SET NO ERROR CONDITION
25650  IF FRNT%(IQ) =  - 1 THEN 25760: REM   IS QUEUE EMPTY ?
25660  REM  RESET FRONT POINTER AND RETRIEVE DATA
25670 PT% = FRNT%(IQ)
25680 FRNT%(IQ) = LINK%(PT%)
25690  REM  IF QUEUE IS EMPTY, SET REAR POINTER
25700 LDTA% = LQDTA%(PT%)
25710  REM  IF QUEUE IS EMPTY, SET REAR POINTER
25720  IF FRNT%(IQ) =  - 1 THEN REAR%(IQ) =  - 1
25730 IDTA = PT%: REM   AVAILABLE LOCATION
25740  GOSUB 24620: REM   PUT NEW AVAILABLE LOCATION ON STACK
25750  RETURN
25760 IL = 1: REM   SET ERROR CONDITION
25770  PRINT : PRINT "QUEUE IS EMPTY;CAN`T DELETE ELEMENT": RETURN
```

Figure 14.11—Linked List Subroutines

After these data have been placed in the respective queues, your program should print out the contents of each queue; delete one data element from queue 1 and queue 3; and

then print out the queue contents. Assume that no more than three queues are required and that the maximum size of the data array is 6.

SOLUTION. Figure 14.12 contains one possible solution to Example 3. You should note that the data needed by the example program (queue number, data values) are supplied in DATA statements at the end of the main program. The maximum number of queues is set to 3, and the size of the data array is set to 6.

```
100  REM  MAIN PROGRAM ILLUSTRATING USING LINKED QUEUES
110 SIZ% = 6: REM                          MAXIMUM SIZE OF ARRAYS
120 NQ% = 3: REM                           MAXIMUM NUMBER OF QUEUES
130  REM  INITIALIZE STACK TO 0
140  GOSUB 24570
150  REM  INITIALIZE LINKED QUEUES TO 0 AND PLACE AVAILABLE
160  REM  SPACE IN STACK
170  GOSUB 25300
180  REM  ADD DATA ELEMENTS TO LINKED QUEUES
190  GOSUB 300
200  REM  PRINT QUEUE CONTENTS
210  GOSUB 470
220  REM  DELETE DATA ELEMENTS FROM LINKED QUEUES
230  GOSUB 390
240  REM  PRINT QUEUE CONTENTS
250  GOSUB 470
260  DATA  1,50,2,75,1,10,3,113,-9999
270  DATA  1,3,-9999
280  END
290  REM  ******************************************************
300  REM  SUBROUTINE: INPUT
310  READ IQ
320  IF IQ =  - 9999 THEN  RETURN
330  IF (IQ < 1) OR (IQ > NQ%) THEN  PRINT "INVALID QUEUE": END
340  READ LDTA%
350  GOSUB 25460: REM                    ADD TO LINKED QUEUE
360  IF IL <  > 0 THEN  PRINT "ERROR IN ADDING TO QUEUE": END
370  GOTO 310
380  REM  ******************************************************
390  REM  SUBROUTINE: REMOVE
400  READ IQ
410  IF IQ =  - 9999 THEN  RETURN
420  IF (IQ < 1) OR (IQ > NQ%) THEN  PRINT "INVALID QUEUE": END
430  GOSUB 25630: REM                    DELETE FROM LINKED QUEUE
440  IF IL <  > 0 THEN  PRINT "ERROR IN DELETING FROM QUEUE": END
450  GOTO 400
460  REM  ******************************************************
470  REM  SUBROUTINE: PRINT LINKED QUEUE CONTENTS
480  PRINT
490  FOR I = 1 TO NQ%
500  PRINT "CONTENTS QUEUE NO. ";I
510  IF FRNT%(I) =  - 1 THEN  PRINT "QUEUE EMPTY": GOTO 570
520 LLOC% = FRNT%(I)
530  PRINT LQDTA%(LLOC%)
540 LLOC% = LINK%(LLOC%): REM              LOCATION OF NEXT ELEMENT IN QUEUE
550  IF LLOC% =  - 1 THEN 570
560  GOTO 530
570  NEXT I
580  PRINT : PRINT
590  RETURN

]
```

Figure 14.12—Linked List Subroutines

Data are read into the program using subroutine INPUT. The data to be input to a particular queue must reside in a DATA statement in the following format: queue number, data value. This subroutine terminates reading data when it encounters the value −9999 for the queue number. With subroutine REMOVE we can specify from what queue a data element is to be removed. The queue number is read from a DATA statement for each element to be removed. When the number −9999 is encountered in the DATA statement, this subroutine terminates reading data and no more data are removed from any of the queues. A subroutine has also been written to print the contents of each queue.

Bidirectional Linked Lists

The linked list structure discussed in the previous section works well with LIFO and FIFO ordered lists. Additions and deletions to these types of lists occur only at the ends. However, many other types of ordered lists may involve additions and deletions somewhere between the ends of the lists. For example, deleting a name from inside an alphabetical list requires changing the pointer that linked this name in the list to the next (successor) name in the list. The successor name is easily found because the pointer to this name is associated with the name being deleted. However, the preceding (predecessor) name must be located. Locating this name requires searching through the list.

Bidirectional lists make random deletions and insertions to linked lists easier. Associated with each data element in a list are two sets of pointers. One set points forward to each *successor* in the list; another set points backward to each *predecessor* in the list. Figure 14.13 illustrates a bidirectional list in which employee names are maintained in alphabetical order by department.

Using Figure 14.13, we can obtain the following lists of employees ordered alphabetically:

Department 1	Department 2
Baker	Apple
Brady	Jones
Hall	Smith

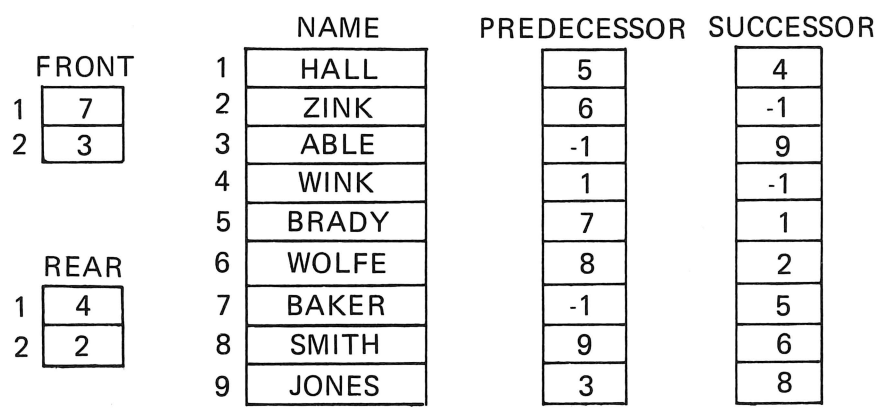


Figure 14.13—Bidirectional Lists

Wink Wolfe
Zink

With the successor and predecessor pointers, this type of list can be processed in the forward or backward direction. Deleting the name Smith, in Department 2, is easy once the name is located:

	Predecessor	Successor
Smith	9	6

From this information, we know that the predecessor to Smith is located in row 9 and the successor is located in row 6. Therefore, the successor pointer for the name in row 9 (Jones) must be changed from 8 to 6 (pointing to Wolfe). Now the predecessor for the name in row 6 (Wolfe) must be changed from 8 to 9 (pointing to Jones). The result of these changes is shown in Figure 14.14.

Assuming that the predecessor and successor pointers are maintained in the respective arrays PRD and SUC, the required changes in a program would be:

SUC%(PRD%(8)) = SUC%(8), giving SUC%(9) = 6; and

PRD%(SUC%(8)) = PRD%(8), giving PRD%(6) = 9.

Consequently, you can see that deleting a name required two statements to make the needed pointer changes. This is much less time consuming than searching a list to find the immediate predecessor to the name deleted, then changing the predecessor pointer.

Figure 14.15 contains programs that utilize a bidirectional list structure. It is assumed that each data element is a string. Previous programs had assumed that all data were integer type. However, the programs in Figure 14.11 can be easily modified to work with other data types (integer and real) by changing the variable declaration from QDTA$ and DTA$ to the desired data type, such as DTA.

These routines initialize the bidirectional list and place all available storage locations on a stack. Additions and deletions can be made to the bidirectional lists. Strings are

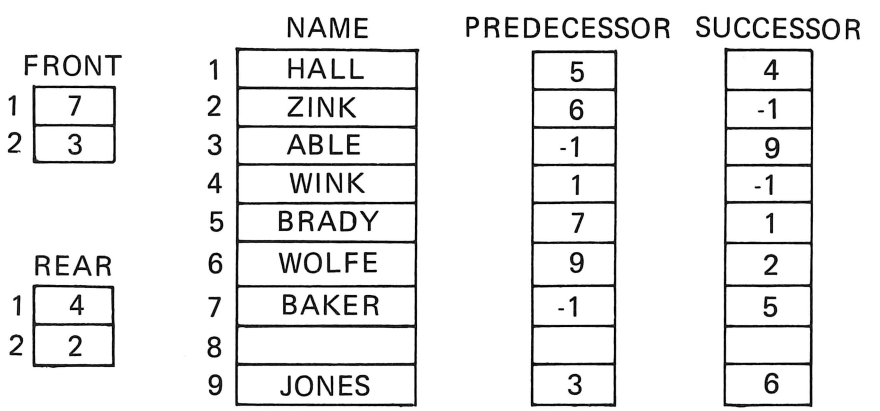


Figure 14.14—Additional Bidirectional List Illustrations

```
25800  REM  SUBROUTINES FOR BIDIRECTIONAL LINKED LISTS
25810  REM  THESE SUBROUTINES WILL INITIALIZE, PERFORM ADDITIONS
25820  REM  AND DELETIONS TO BIDIRECTIONAL LINKED LISTS. THE STRING
25830  REM  RESIDES IN AN ARRAY CALLED QDTA$; FORWARD POINTERS
25840  REM  RESIDES IN AN ARRAY PRD%; BACKWARD POINTERS RESIDE
25850  REM  IN THE ARRAY SUC%. FRNT%(I) POINTS TO FRONT OF LIST
25860  REM  I; REAR%(I) POINTS TO REAR OF LIST I. LOCATION IN ARRAY
25870  REM  QDTA$ WHERE DATA IS TO BE ADDED OR DELETED IS INDICATED
25880  REM  BY PT%. VARIABLE DTA$ CONTAINS STRING TO BE ADDED OR
25890  REM  STRING THAT WAS REMOVED. STACK ARRAY SK CONTAINS A
25900  REM  LIST OF AVAILABLE SPACE IN THE DATA ARRAY.
25910  REM  *****************************************************
25920  REM  SUBROUTINE: INITIALIZE BIDIRECTIONAL LISTS
25930  DIM QDTA$(SIZ%),PRD%(SIZ%),SUC%(SIZ%)
25940  DIM FRNT%(NQ%),REAR%(NQ%)
25950  FOR I = 1 TO NQ%
25960 FRNT%(I) =  - 1: REM  DENOTE THAT LIST ARE EMPTY
25970 REAR%(I) =  - 1
25980  NEXT I
25990  FOR I = SIZ% TO 0 STEP  - 1: REM  LOAD STACK WITH AVAILABLE LOCATIONS
26000 IDTA = I
26010  GOSUB 24620: REM  PUSH AVAILABLE LOCATION ON STACK
26020  NEXT I
26030  RETURN
26040  REM  *****************************************************
26050  REM  SUBROUTINE: ADD TO A BIDIRECTIONAL LIST
26060 IL = 0: REM  SET NO ERROR CONDITION
26070  GOSUB 24710: REM  OBTAIN AVAILABLE DATA CELL
26080  IF IL <  > 0 THEN  PRINT "ERROR STACK ROUTINE": END
26090 PT% = IDTA: REM  PT% IS AVAILABLE DATA CELL
26100  IF (PT% < 0) OR (PT% > SIZ%) THEN 26180: REM  DOES LOCATION EXIST?
26110  IF (FRNT%(IQ) =  - 1) THEN 26190: REM  LIST IS EMPTY
26120 SEQ% = FRNT%(IQ)
26130  IF DTA$ < QDTA$(SEQ%) THEN 26240: REM  INSERT AT FRONT
26140  IF REAR%(IQ) = SEQ% THEN 26300: REM  INSERT AT REAR OF LIST
26150 SEQ% = SUC%(SEQ%)
26160  IF DTA$ < QDTA$(SEQ%) THEN 26360: REM  INSERT IN LIST
26170  GOTO 26140
26180  PRINT : PRINT "INVALID LOCATION FOR DATA ELEMENT": END
26190 FRNT%(IQ) = PT%: REM  EMPTY LIST, INSERTION AT FRONT
26200 REAR%(IQ) = PT%: REM  EMPTY LIST, INSERTION AT FRONT
26210 PRD%(PT%) =  - 1:SUC%(PT%) =  - 1
26220 QDTA$(PT%) = DTA$
26230  RETURN
26240 SUC%(PT%) = FRNT%(IQ): REM  INSERTION AT FRONT OF LIST
26250 PRD%(PT%) =  - 1
26260 PRD%(FRNT%(IQ)) = PT%
26270 FRNT%(IQ) = PT%
26280 QDTA$(PT%) = LDTA$
26290  RETURN
26300 SUC%(SEQ%) = PT%: REM  INSERTION AT FRONT OF LIST
26310 SUC%(PT%) =  - 1
26320 PRD%(PT%) = SEQ%
26330 REAR%(IQ) = PT%
26340 QDTA$(PT%) = DTA$
26350  RETURN
26360 SUC%(PRD%(SEQ%)) = PT%: REM  INSERTION IN LIST
26370 PRD%(PT%) = PRD%(SEQ%)
26380 PRD%(SEQ%) = PT%
26390 SUC%(PT%) = SEQ%
26400 QDTA$(PT%) = DTA$
26410  RETURN
26420  REM  *****************************************************
26430  REM  SUBROUTINE: DELETE STRING FROM BIDIRECTIONAL LIST
26440  REM  WILL DELETE FIRST, MATCHING, OR LAST STRING FROM LIST.
26450  REM       ITY = 1 DELETE FIRST STRING
```

```
26460  REM        ITY = 2 DELETE MATCHING STRING
26470  REM        ITY = 3 DELETE LAST STRING
26480 IL = 0: REM  SET NO ERROR CONDITION
26490  IF FRNT%(IQ) =  - 1 THEN 26860: REM  IS LIST EMPTY?
26500  ON ITY GOTO 26510,26710,26630
26510  REM  DELETE FIRST STRING FROM LIST
26520 SEQ% = FRNT%(IQ)
26530 DTA$ = QDTA$(SEQ%)
26540  IF REAR%(IQ) < > SEQ% THEN 26600: REM  IS LIST EMPTY?
26550 FRNT%(IQ) =  - 1:REAR%(IQ) =  - 1: REM  LIST IS EMPTY
26560 IDTA = SEQ%
26570  GOSUB 24620: REM  PUT AVAILABLE SPACE ON STACK
26580  IF IL < > 0 THEN  PRINT "ERROR IN STACK ROUTINE": END
26590  RETURN
26600 FRNT%(IQ) = SUC%(SEQ%): REM  LIST IS NOT EMPTY
26610 PRD%(SUC%(SEQ%)) =  - 1
26620  GOTO 26560
26630  REM  DELETE LAST STRING FROM BIDIRECTIONAL LIST
26640 SEQ% = REAR%(IQ)
26650 DTA$ = QDTA$(SEQ%)
26660  IF FRNT%(IQ) < > SEQ% THEN 26680: REM  IS LIST EMPTY?
26670  GOTO 26550: REM  EMPTY LIST, PUSH LOCATION ON STACK
26680 REAR%(IQ) = PRD%(SEQ%): REM  LIST IS NOT EMPTY
26690 SUC%(PRD%(SEQ%)) =  - 1
26700  GOTO 26560: REM  PUSH LOCATION ON STACK
26710  REM  DELETE MATCHING STRING FROM BIDIRECTIONAL LIST
26720 SEQ% = FRNT%(IQ)
26730  IF DTA$ = QDTA$(SEQ%) THEN 26770: REM  MATCH FOUND
26740  IF REAR%(IQ) = SEQ% THEN 26830: REM  NO MATCH FOUND
26750 SEQ% = SUC%(SEQ%)
26760  GOTO 26730
26770  REM  DELETE STRING
26780  IF SEQ% = FRNT%(IQ) THEN 26510: REM  GO TO DELETE FIRST STRING
26790  IF SEQ% = REAR%(IQ) THEN 26630: REM  GO TO DELETE LAST STRING
26800 SUC%(PRD%(SEQ%)) = SUC%(SEQ%)
26810 PRD%(SUC%(SEQ%)) = PRD%(SEQ%)
26820  GOTO 26560: REM  PUSH LOCATION ON STACK
26830  REM  NO MATCH FOUND
26840  PRINT : PRINT "NO MATCH FOUND FOR STRING:",DTA$
26850  RETURN
26860 IL = 1: REM  SET ERROR CONDITION
26870  PRINT : PRINT "CANNOT REMOVE AN ELEMENT-LIST IS EMPTY": END
```

Figure 14.15—Bidirectional List Subroutines

maintained in alphabetical order. However, deletions from a specified list can be made in three different ways depending upon the value of the control parameter ITY:

ITY = 1, delete the first string.
ITY = 2, delete the matching string.
ITY = 3, delete the last string.

The value for ITY and the matching string are supplied by the main program. In addition, the number of lists (NQ) and the maximum size (SIZ) of the arrays are supplied by the main program.

EXAMPLE 4. Using a bidirectional list, store the following names in the respective lists:

List 1	List 2
Baker	Able
Brady	Jones
Hall	Wolfe
Wink	Zink

After loading the lists, print the contents of each list. Then delete the first string (name) from List 1, the last string from List 2, and a string matching Wolfe from List 2. Now, print the contents of the remaining lists.

SOLUTION. Figure 14.16 contains a program that provides one possible solution to Example 4. This program has been written using subroutines (INPUT, REMOVE, etc.) like the solution to Example 3. The INPUT subroutine expects data to be in a specific format:

string, list number

The DATA statements in lines 550 and 560 conform to this format. Thus:

BAKER, 1

means that the string BAKER is to be placed in list 1. Also, the value −9999 will terminate input. This value must be in the position a string would occupy in the DATA statement.

The DATA statement in line 570 contains information that specifies what types of deletions are to occur. The following format should be used for deletion types 1 (delete first string) and 3 (delete last string):

deletion type, list number

If deletion type 2 (matching string) is specified, the following format should be used:

deletion type, list number, matching string

The value −9999 appearing in the deletion type field will terminate the deletions.

14.5 Summary

In this chapter, we discussed several types of data structures. Familiarity with these structures can be very beneficial when you are developing your own programs. Subroutines were provided that will aid you in implementing structures such as stacks, queues, linked queues, and bidirectional linked queues. Standish (1980) describes other structures not discussed here, such as trees. Although the techniques presented in this chapter were illustrated using arrays in memory, many of the techniques can be used to structure disk files. For example, consider the bidirectional linked lists appearing in Figure 14.13. This structure can be implemented in a disk file, substantially reducing the time needed to access these records by department. Each record would contain a pointer field that holds the number of the next record in the list. Thus, linked lists and other data structure techniques can be applied to disk files. A set of programs called data base management systems are used to manage data structures in disk files. Systems of this type are widely used on minicomputers and mainframe computers. Some data base management systems

```
100  REM  MAIN PROGRAM ILLUSTRATING BIDIRECTIONAL LINKED LISTS
110  REM  STRINGS WILL BE STORED IN LISTS
120 SIZ% = 12: REM                          MAXIMUM SIZE ARRAYS
130 NQ% = 3: REM                            MAX NUMBER OF LISTS
140  REM  INITIALIZE STACK TO 0
150  GOSUB 24570
160  REM  INITIALIZE LISTS TO BLANK AND PLACE AVAILABLE SPACE IN STACK
170  GOSUB 25880
180  REM  ************************************************
190  REM  SUBROUTINE: INPUT
200  READ DTA$
210  IF DTA$ = "-9999" THEN 250: REM      END OF DATA
220  READ IQ: REM                        LIST NUMBER
230  GOSUB 26050: REM                    GO TO ADD SUBROUTINE
240  GOTO 200: REM                       READ NEXT DATA
250  REM  PRINT CONTENTS OF LISTS
260  GOSUB 430: REM                      GO TO PRINT SUBROUTINE
270  PRINT : INPUT "PRESS RETURN TO CONTINUE -> ";AN$
280  REM  ************************************************
290  REM  SUBROUTINE: DELETE
300  READ ITY: REM                       TYPE OF DELETION
310  IF ITY =  - 9999 THEN 390: REM         END OF DATA
320  IF ITY = 2 THEN 360
330  READ IQ: REM                        LIST NUMBER
340  GOSUB 26430: REM                    GO TO DELETE SUBROUTINE
350  GOTO 300: REM                       READ NEXT DATA
360  READ IQ,DTA$: REM                   LIST NUMBER,STRING MATCH
370  GOSUB 26430
380  GOTO 300: REM                      READ NEXT DATA
390  REM  PRINT ELEMENTS OF LISTS
400  GOSUB 430: REM                      GO TO PRINT SUBROUTINE
410  GOTO 580
420  REM  ************************************************
430  REM  SUBROUTINE: PRINT BIDIRECTIONAL LIST CONTENTS
440  PRINT
450  FOR I = 1 TO NQ%
460  PRINT : PRINT "CONTENTS OF LIST NO. ";I: PRINT
470  IF FRNT%(I) =  - 1 THEN  PRINT "LIST EMPTY": GOTO 530
480 LLOC% = FRNT%(I)
490  PRINT QDTA$(LLOC%)
500 LLOC% = SUC%(LLOC%)
510  IF LLOC% =  - 1 THEN 530: REM          END OF LIST?
520  GOTO 490
530  NEXT I
540  RETURN
550  DATA  BAKER,1,ABLE,2,JONES,2,BRADY,1,HALL,1
560  DATA  SMITH,2,WINK,1,WOLFE,2,-9999
570  DATA  1,1,3,2,2,2,WOLFE,-9999
580  END

]
```

Figure 14.16—Using Bidirectional List Subroutines

are becoming available for microcomputers. Bradley (1981) provides a discussion of the concepts used to develop a data base management system.

References

Bradley, James, *File and Data Base Techniques*, Holt Rinehart and Winston, New York, 1981.
Standish, Thomas A., *Data Structure Techniques*, Addison-Wesley, Reading, Ma., 1980.

Exercises

1. What is a stack structure?
2. What is a queue data structure?
3. Assume that A is a single-dimensional array [DIM A(50)]. You are to use a stack to manage the available storage space in this array. The largest array elements are to be used first [A(50), A(49), etc.]. As an element becomes available to be used to store data, that element is to be "pushed" onto the stack. Write a program that will satisfy the specifications of this exercise. You are also to use the stack subroutine provided in this chapter.
4. A series of tests are to be performed over an extended time period. Each time a test is performed, a numeric value in the range of 0 to 499.99 is recorded. Altogether, 200 tests are to be performed. However, only the most recent 25 readings will be maintained. Therefore, this data will be maintained on a FIFO (first-in, first-out) basis. After more than 25 test results are compiled, the oldest result will be discarded as the new result is saved. Write a program which will store this data in an array having no more than 25 elements. If possible, use subroutines from this chapter in developing this program.
5. The following data must be saved each time an inspection is performed on a product prior to shipping.

   inspector's name
   date
   value 1
   value 2
   value 3

   You are to write a program which will maintain 100 sets of inspection results in memory. All results for an inspector are to be linked together with a bidirectional linked list. There are 4 different inspectors.
6. Write a program which stores the data from exercise 5 in a random access disk file. Each record in the file should be bidirectionally linked.
7. Sometimes an NXM array must be sorted using one of the columns as a "key" for the sort procedure. For instance, a 50×3 array (50 rows $\times$ 3 columns) may contain an individual's social security number in column 1, his age in column 2, and his weight in column 3. Such an array might contain data for 50 people (50 rows). We might want to sort this data on age. Using the routines provided in this chapter can mean moving all data in a row when an age must be moved to another location in the array. A better way is to use a linked list where only the pointers in the list are changed. You are to modify the bubble sort algorithm presented in Chapter 12 so that an array can be sorted using any column as the key. You are also to use a linked list to implement your algorithm.

15

Random Numbers & Simulation

Applesoft BASIC has a numeric function named RND which can generate "random" numbers between 0 and 1. Each number in this interval is equally likely to occur and successively generated values are independent. The RND function can be used for such things as games, simulation models, and statistical studies. Every time RND is executed, a number between 0 and 1 is generated. The number generated is called a *pseudo-random number* because an algorithm is used to generate it. This is not equivalent to performing an experiment such as throwing a die to generate a number between 1 and 6. However, functions like RND permit us to generate numbers that emulate random numbers.

15.1 Using The Function RND

The RND function has the following format:

RND(n)

where n is a numeric constant, variable, or expression. The value returned by RND depends on the argument n. If n is negative, RND always starts a repeatable sequence of random numbers. After the initial use of a negative argument to generate a repeatable sequence, a positive argument should be used to generate the remaining numbers in the sequence. Different negative arguments will result in different sequences of numbers. If n is positive, RND returns a different number each time unless a repeatable sequence has been initiated with a negative argument. If n is 0, RND returns the random number most recently generated. We will present some examples to illustrate the use of RND.

The algorithm used in RND utilizes the last random number generated to generate the next random number. Consequently, by specifying the argument n for the first random number, we are in a way controlling all random numbers that will be generated until the n is reset. This will also permit us to generate a given random number sequence as many times as desired. Control such as this can be very useful in validating a simulation program or other computer programs using random numbers. Consider the following program:

```
10 FOR I = 1 TO 5
20 PRINT RND(100)
30 NEXT I
40 END
```

After running this program, five random numbers will be displayed:

```
.280111176
.836328912
.115622079
.815110127
.57325385
```

Everytime you run the above program you will get a different sequence of numbers. Now change the program so that the first time RND is used the argument is negative:

```
 5 PRINT RND (-100)
10 FOR I=1 TO 4
20 PRINT RND(100)
30 NEXT I
40 END
```

When you run this program. The following numbers will be displayed.

```
4.66889105E-08
 .402299377
 .249404703
 .960296315
 .640900414
```

The same random numbers will be displayed each time this program is run using −100 as the initial argument of RND.

A random number between 0 and 1 will not always satisfy our needs. For example, we may be writing a program that simulates rolling a die. If a die is unbiased, throwing it several times will result in a series of uniformly distributed integers in the range of 1 to 6. Consequently, we need to be able to generate a random integer having the value of 1, 2, 3, 4, 5, or 6.

The function RND generates a random number between 0 and 1. Specifically, the random X is in the range,

$$0 \leq X < 1$$

Therefore, the expression

```
X = RND(100)*6
```

will generate a number in range of

$$0 \leq X < 6.$$

Now we must convert these numbers to integers:

```
I = INT(RND (100)*6),
```

This expression produces an integer in the range of

$$0 \leq I \leq 5.$$

Each integer in this range has an equal likelihood of occurring. Consequently, if we add 1 to the above expression we can simulate rolling a die:

```
I = INT(RND(100)*6) + 1
```

EXAMPLE 1. Write a program that will simulate tossing two dice. After each toss, display the value for each die and the total of the dice. Write your program so that you can control how many times the dice are tossed.

SOLUTION. In the following program, the variables D1 and D2 represent the individual die.

```
10 D1 = INT(RND(1)*6) + 1
20 D2 = INT(RND(1)*6) + 1
30 TTAL = D1 + D2
40 PRINT "DIE 1 "; D1, "DIE 2 "; D2
50 PRINT "TOTAL "; TTAL
60 INPUT "ROLL AGAIN (Y/N)?"; Y$
70 IF Y$="Y" THEN PRINT:GOTO 10
80 END
```

You might run this program several times using different initial arguments for RND to determine if the integers generated for the individual die are uniformly distributed between 1 and 6.

15.2 Developing a Random Number Generator

We will describe one of the most popular methods of generating pseudo-random numbers, the congruential method, in this section. The interested reader might refer to Shannon (1975) for additional material on this subject. The congruential method involves multiplying a constant, C, by another number, L, that varies in value, to obtain a product. Another number, R (called the residual), is created from the value of the product by keeping the last x digits of the product. An expression for this process is:

$$R = L*C - M*INT(L*C/M),$$

where R = residual

C = constant multiplier

L = varying multiplier

M = modulus (used to obtain residue)

Consider an example where we want to keep the last two digits of the product, L*C; therefore, we will set M = 100. Also, let C = 33 and L = 13. Using the above expression,

$$\begin{aligned} R &= 13*33 - 100*INT(13*33/100) \\ &= 429 - 100*INT(4.29) \\ &= 429 - 100*4 \\ &= 429 - 400 \\ &= 29 \end{aligned}$$

The product of C times L is 429. Keeping the residual, composed of the last two digits, we get 29.

The next number in the sequence would be generated by replacing the old value of L, 13, with the new residual, 29. Using this value for L and substituting into the above expression, we get:

$$R = 29*33 - 100*\text{INT}(29*33/100)$$

$$= 57$$

Continuing this process until we have generated ten numbers results in the following pseudo-random number series:

29,57,82,73,9,97,1,33,89,37

If these numbers are all divided by M, the resulting series is between 0 and 1. You should be careful in choosing the values for C and M. Experience has shown that M should be large, C should not be too small relative to M, and M and C should have no common factors.

Figure 15.1 contains a subroutine called ORGRND, which will generate random numbers using the congruential method. We selected M = 8,388,608 (2^{23}) and C = 2,893 (an odd number approximately the square root of 2^{23}). To use this subroutine, you must specify a seed in the main program. The seed must be negative and within the range $-8388608 \leq \text{SEED} \leq 0$. You may reseed the generator at any time by specifying a negative seed.

```
27000  REM  ***********************************************
27010  REM  SUBROUTINE: ORGRND RANDOM NUMBER GENERATOR
27020  REM  CONGRUENTIAL METHOD
27030  REM  USER MUST INITIALLY SPECIFY SEED (-8388608<SEED<0)
27040  REM  IN MAIN PROGRAM, SETTING SEED EQUAL TO A NEGATIVE
27050  REM  VALUE WILL RESEED GENERATOR.
27060 M = 8388608: REM    MODULUS VALUE
27070 C = 2893: REM       CONSTANT MULTIPLIER
27080  IF SEED > 0 THEN 27140
27090  IF SEED = 0 THEN L = C:SEED = L: GOTO 27140
27100  REM  CHECK FOR VALID SEED
27110  IF (SEED <  - 8388608) THEN  PRINT "INVALID SEED": END
27120 SEED = ( - 1) * SEED
27130 L = SEED
27140 L = L * C - M *  INT (L * C / M)
27150 RND = L / M
27160  RETURN
```

Figure 15.1—ORGRND Random Number Generator

15.3 Testing a Random Number Series

It is possible to statistically check a series of numbers to determine if they are uniformly distributed. One such test is called a *chi-square goodness of fit test*. To apply this test, we must divide the range of the numbers generated into I equal intervals. A count is then made of the random numbers that fall within each interval. If the numbers are uniformly distributed, the count for each interval should be approximately the same.

Using the function RND, we can generate random numbers in the range of 0 to 1, Figure 15.2 consists of a program which will generate such a series.

```
100 REM PROGRAM RANDOM NUMBERS USING RND
110 REM ARRAY AY CONTAINS RANDOM NUMBER SERIES.
120 REM N IS THE LENGTH OF THE SERIES.
130 INPUT "ENTER INITIAL VALUE FOR RND ARGUMENT ";X
135 X=RND(X):REM ASSUME INITIAL ARGUMENT IS NEGATIVE
140 INPUT "ENTER NUMBER FOR LENGTH OF SERIES"; N
150 DIM AY(N)
160 PRINT:PRINT "GENERATING ";N; " RANDOM NUMBERS"
170 FOR I=1 TO N
180 RNO=RND(1)
190 AY(I)=RNO
200 NEXT I
220 END
```

Figure 15.2—Generating Random Numbers

A natural way to segment this range is to specify 10 intervals. Thus the first interval would be from 0 to .1. A count would be made of all the numbers that had a value within this range. The same procedure would be followed for the interval of .1 to .2, etc. We would expect the count in each of the intervals to be equal to N/10, where there are N numbers.

Having determined the count for each interval, the chi-square statistic S can be determined:

$$S = \frac{(C(1)-A)^2}{A} + \frac{(C(2)-A)^2}{A} + \cdots + \frac{(C(10)-A)^2}{A}$$

where C(1) is the count for the first interval and A = N/10. When 10 intervals are used and $S \geq 16.92$, we can be 95% confident that the series is not uniformly distributed. In that case, our generator would need some improvements. The value of 16.92 is only valid for a test involving 10 intervals. This value was obtained from a table of percentage points for the chi-square distribution; 16.92 is a chi-square value for the 95 percentile with 9 degrees of freedom. Several other statistical tests can be used to evaluate the random behavior of a series of pseudo-random numbers. Shannon (1975) contains a brief summary of the most common tests applied.

In Figure 15.3 you will find a subroutine that will perform a chi-square goodness of fit test on a series of numbers. The main program must supply the series in an array, AY, and the length of the series, N. The program uses 10 intervals and compares the chi-square statistic to 16.9.

EXAMPLE 2, PART A. Using the chi-square goodness of fit test subroutine in Figure 15.3, test a series of 100 numbers generated using the RND function.

PART B. Likewise, test a series of 100 numbers generated using the ORGRND subroutine in Figure 15.1.

SOLUTION, PART A. The program in Figure 15.4 generates 100 random numbers using RND and then calls the chi-square goodness of fit subroutine. A random number seed of −1 was used to get the results shown in Figure 15.5.

```
27200  REM  ******************************************************
27210  REM  SUBROUTINE: CHI-SQUARE GOODNESS OF FIT TEST
27220  REM  FOR A UNIFORMLY DISTRIBUTED SERIES OF
27230  REM  NUMBERS IN THE RANGE OF 0<X<1.
27240  REM  ARRAY AY CONTAINS A SERIES OF NUMBERS.
27250  REM  N IS THE LENGTH OF THE SERIES
27260  DIM C(10)
27270 CHI = 16.9
27280 K = 10
27290  FOR I = 1 TO N
27300 INV = 0
27310  FOR J = 1 TO K
27320  IF (AY(I) > INV) AND (AY(I) <  = (INV + .1)) THEN 27360
27330 INV = INV + .1
27340  NEXT J
27350  PRINT : PRINT "NUMBER OUT OF RANGE": END
27360 C(J) = C(J) + 1
27370  NEXT I
27380 S = 0:A = N / K
27390  PRINT
27400  PRINT : PRINT  TAB( 7)"RESULTS OF CHI-SQUARE TEST": PRINT :
27410  PRINT "INTERVAL   ACTUAL  EXPECTED  CHI-SQUARE"
27420  PRINT " NUMBER    OBSER.   OBSER.    COMPONENT"
27430  FOR J = 1 TO K
27440 S1 = ((C(J) - A) ^ 2) / A
27450 S1 =  INT (S1 * 10 ^ 3 + .5) /  INT (10 ^ 3 + .5)
27460  PRINT  TAB( 4);J; TAB( 14);C(J); TAB( 24);A; TAB( 35);S1
27470 S = S + S1
27480  NEXT J
27490  PRINT
27500  PRINT "CHI-SQUARE SUM=";S
27510  IF S >  = CHI THEN 27550
27520  PRINT : PRINT "WE CANNOT REJECT THE HYPOTHESIS THAT"
27530  PRINT "THE NUMBER SERIES IS UNIFORMLY
27540  PRINT "DISTRIBUTED"
27550  RETURN
27560  PRINT "WE ARE 95% CONFIDENT THAT THE NUMBER"
27570  PRINT "SERIES IS NOT UNIFORMLY DISTRIBUTED."
27580  RETURN
```

Figure 15.3—Chi-Square Goodness of Fit Subroutine

```
100 REM PROGRAM TO TEST RANDOM NUMBER GENERATOR
110 REM ARRAY AY CONTAINS NUMBER SERIES
120 REM N IS THE LENGTH OF THE SERIES
130 DIM AY(500)
140 INPUT "ENTER INITIAL VALUE RND ARGUMENT ";X
145 X=RND(X)
150 INPUT "ENTER LENGTH OF RANDOM NUMBER SERIES";N
160 PRINT:PRINT "GENERATING ";N;" RANDOM NUMBERS"
170 FOR I = 1 TO N
180 RNO=RND(1)
190 AY(I)=RNO
200 NEXT I
210 GOSUB 27210
220 END
```

Figure 15.4—Generating 100 Numbers Using RND

RESULTS OF CHI-SQUARE TEST

Interval Number	Actual Observations	Expected Observations	Chi-Square Components
1	16	10	3.6
2	13	10	.9
3	7	10	.9
4	4	10	3.6
5	9	10	.1
6	9	10	.1
7	10	10	0
8	11	10	.1
9	10	10	0
10	11	10	.1

Chi-Square Sum = 9.4

Figure 15.5—Chi-Square Results, RND Function

PART B. Figure 15.6 contains a program which performs a similar analysis using the ORGRND subroutine. Note that only three statements were changed (statements 140, 145, and 180). A seed of −1 was used to get the results shown in Figure 15.7. From these results, we cannot reject the hypothesis that RND and ORGRND generate random numbers in the interval of 0 to 1.

```
100 REM PROGRAM TO TEST RANDOM NUMBER GENERATOR
110 REM ARRAY AY CONTAINS NUMBER SERIES
120 REM N IS THE LENGTH OF THE SERIES
130 DIM AY(500)
140 INPUT "ENTER RANDOM NUMBER SEED (-8388608 TO 0)";SEED
145 IF SEED>0 THEN PRINT "INVALID":GOTO 140
150 INPUT "ENTER LENGTH OF RANDOM NUMBER SERIES";N
160 PRINT:PRINT "GENERATING ";N;" RANDOM NUMBERS"
170 FOR I = 1 TO N
180 GOSUB 27000
190 AY(I)=RNO
200 NEXT I
210 GOSUB 27210
220 END
```

Figure 15.6—Generating 100 Numbers Using ORGRND

RESULTS OF CHI-SQUARE TEST

Interval Number	Actual Observations	Expected Observations	Chi-Square Component
1	10	10	0
2	6	10	1.6
3	8	10	.4
4	16	10	3.6
5	10	10	0
6	11	10	.1
7	6	10	1.6
8	6	10	1.6
9	11	10	.1
10	16	10	3.6

Chi-Square Sum = 12.6

Figure 15.7—Chi-Square Results, ORGRND Subroutine

15.4 Random Number Generators for Specific Distributions

Computers are often used to simulate real world activities such as the services of a bank or a particular inventory replenishment policy. Simulation is often used because a mathematical model of the physical process may be too complicated to solve; therefore, the physical activities might be modeled using a computer program. An example of a simple simulation model will be presented in the next section. Another reason for using simulation is to introduce uncertainty (randomness) into the analysis. Analytical solutions to a model may only give unexpected values (averages); simulation studies can provide additional insight into the variation about the expected value.

One way of introducing uncertainty into simulation studies is by means of random number generators. However, to do this we need to be able to generate random numbers from several types of distributions. Subroutines will be presented in this section that generate random numbers from some of the most commonly used distributions. A main program should set the initial value for the RND argument before using these subroutines, because the RND function is used in each subroutine.

Each of these subroutines was tested by generating a sample of 500 numbers using a random number seed of -1000. Then for each sample, the mean and standard deviation were calculated, and a frequency histogram was plotted. These calculations and plots were made using the subroutines provided in Chapter 2 (Data Reduction). The results are included in the description of each random number generator.

Uniform Distribution

The uniform distribution has a continuous probability density function over some interval, a to b. This distribution is described by:

$$f(x) = \frac{1}{b-a}$$

where the mean is $(b+a)/2$ and the variance is $(b-a)^2/12$.

Since the pseudo-random number generator produces numbers uniformly distributed in the interval of 0 to 1, it is a simple matter to transform these numbers to the interval a to b:

$$x = a+(b-a)*RND(1), \qquad 0 \leq RND < 1$$

Figure 15.8 represents a subroutine that will generate a random number in the interval A to B. Values for A and B must be supplied by the main program. The random number generated resides in the variable RUFM. Figure 15.9 contains a frequency histogram of a sample of 500 numbers generated using this subroutine.

```
28000  REM  ******************************************************
28010  REM  SUBROUTINE: UNIFORM DISTRIBUTION GENERATOR
28020  REM  MAIN PROGRAM SHOULD INITIALLY RANDOMIZE SEED
28030  REM  MAIN PROGRAM MUST SUPPLY VALUES FOR LIMITS OF
28040  REM  INTERVAL, A AND B
28050  REM  RANDOM NUMBER GENERATED IS RUFM
28060 RUFM = A + (B - A) * RND (1)
28070  RETURN
```

Figure 15.8—Uniform Distribution Generator

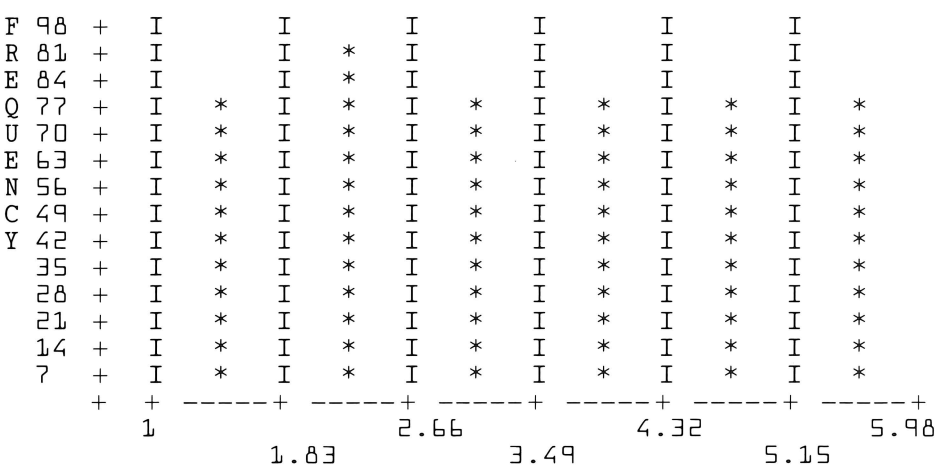


Figure 15.9—Frequency Histogram Uniform Distribution A=1, B=6, N=500

Normal Distribution

The normal (Gaussian) distribution may be the most widely used distribution. It is a continuous distribution characterized by its mean and its variance. This distribution is symmetrical about its mean. To simplify generation of normally distributed numbers, the distribution is usually transformed to have a mean of 0 and a variance of 1. The transform is

$$z = (x-u)/\sigma$$

where u is the mean and σ is the standard deviation. After a standard normal deviate z is generated, it is transformed to the desired normal distribution with the desired mean and variance using

$$x = u + z\sigma.$$

Using the standard normal transform, the normal distribution is described by

$$f(z) = \frac{e^{-z^2/2}}{2\pi}$$

Several methods have been devised to generate normal variates. The one used in the subroutine in Figure 15.10 was proposed by Marsaglia and Bray (1964). Values for the mean, XM, and the standard deviation, SD, must be supplied by the main program. The variable RNR contains the random number generated. A frequency histogram of a sample of 500 numbers appears in Figure 15.11.

```
28080  REM  ******************************************************
28090  REM  SUBROUTINE: NORMAL DISTRIBUTION GENERATOR
28100  REM  MAIN PROGRAM SHOULD INITIALLY RANDOMIZE SEED
28110  REM  MAIN PROGRAM MUST SUPPLY MEAN, XM,
28120  REM  AND STANDARD DEVIATION, SD
28130  REM  RANDOM NUMBER GENERATED IS RNR.
28140  IF NRN = 1 THEN 28240
28150 A1 = 2 *  RND (1) - 1
28160 A2 = 2 *  RND (1) - 1
28170 S = A1 * A1 + A2 * A2
28180  IF S > = 1 THEN 28150
28190 R1N = A1 * SQR (( - 2 *  LOG (S)) / S)
28200 R2N = A2 * SQR (( - 2 *  LOG (S)) / S)
28210 RNR = XM + R1N * SD
28220 NRN = NRN + 1
28230  RETURN
28240 RNR = XM + R2N * SD
28250 NRN = 0
28260  RETURN
```

Figure 15.10—Normal Distribution Generator

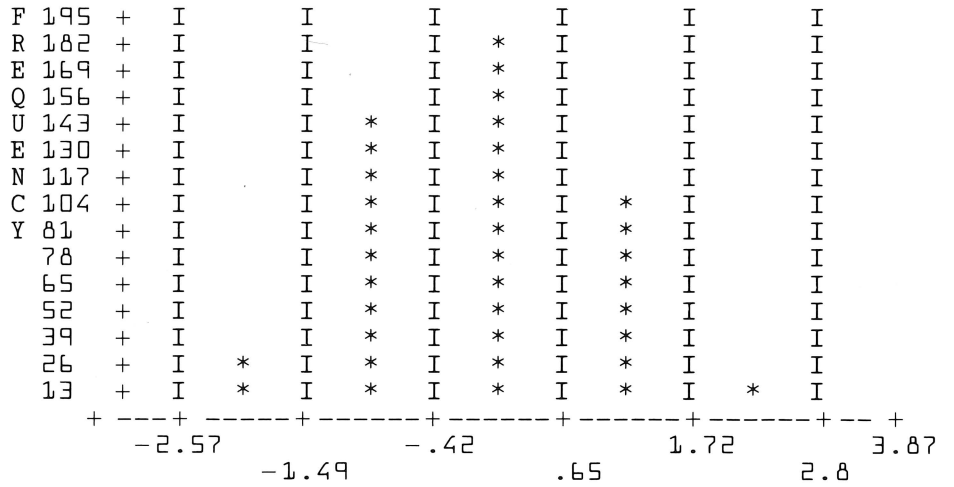


Figure 15.11—Frequency Histogram Normal Distribution $\mu = 0$, $\sigma = 1$, N = 500

Poisson Distribution

The Poisson distribution is often used in statistical and computer simulation studies. This distribution is described by:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

where the mean is λ, the variance is λ, and x is an integer value ≥ 0.

The probability function f(x) describes the probability that an event occurs x times in some interval of time. Consequently, x must be a positive integer value. This distribution might be used to describe the number of arrivals to a bank or the number of car accidents per hour.

The subroutine in Figure 15.12 will generate Poisson variates (for an additional explanation see Reference [3]). The main program should initialize the mean of the distribution, XM. The random number generated resides in the variable RPOI. Figure 15.13 contains a frequency histogram of a sample of 500 numbers.

```
28270  REM  ******************************************************
28280  REM  SUBROUTINE: POISSON DISTRIBUTION GENERATOR
28290  REM  MAIN PROGRAM SHOULD INITIALLY RANDOMIZE SEED
28300  REM  MAIN PROGRAM MUST SUPPLY MEAN, XM
28310  REM  RANDOM NUMBER GENERATED IS RPOI
28320 RPOI = 0
28330 E =  EXP ( - XM)
28340 R = 1
28350 URN =  RND (1)
28360 R = R * URN
28370  IF R < E THEN  RETURN
28380 RPOI = RPOI + 1
28390  GOTO 28350
```

Figure 15.12—Poisson Distribution Generator

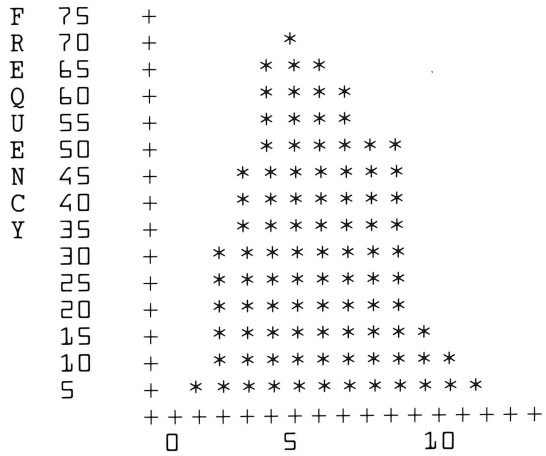


Figure 15.13—Frequency Histogram Poisson Distribution μ = 6, N = 500

Exponential Distribution

The probability is large that no phone calls will occur in the next minute, if the average number of phone calls a day is five. However, the probability is small that no phone calls will occur during the day. If the phone calls are independent, the time between phone calls can be represented by an exponential distribution. This distribution, which is continuous, is described by

$$f(x) = \lambda e^{-\lambda x}$$

where the mean is $1/\lambda$ and the variance is $1/\lambda^2$.

Computer simulation studies often use this distribution because it can represent so many types of phenomena. Some of these are: time between arrival of customers, time between occurrence of automobile wrecks, and life of electronic parts. Figure 15.14 consists of a subroutine that can be used to simulate an exponential distribution. The main program must specify the mean, XM. The variable REXP contains the sample generated. Figure 15.15 contains a frequency histogram of a sample of 500 numbers.

```
28400  REM  ****************************************************
28410  REM  SUBROUTINE: EXPONENTIAL DISTRIBUTION GENERATOR
28420  REM  MAIN PROGRAM SHOULD INITIALLY RANDOMIZE SEED
28430  REM  MAIN PROGRAM MUST SUPPLY MEAN, XM
28440  REM  RANDOM NUMBER GENERATED IS RXP
28450 URN =  RND (1)
28460 RXP =  - XM *  LOG (URN)
28470  RETURN
```

Figure 15.14—Exponential Distribution Generator

Erlang Distribution

If the random numbers must be positive and generally follow a unimodal distribution, then the phenomenon might be represented by an Erlang distribution. The density function is:

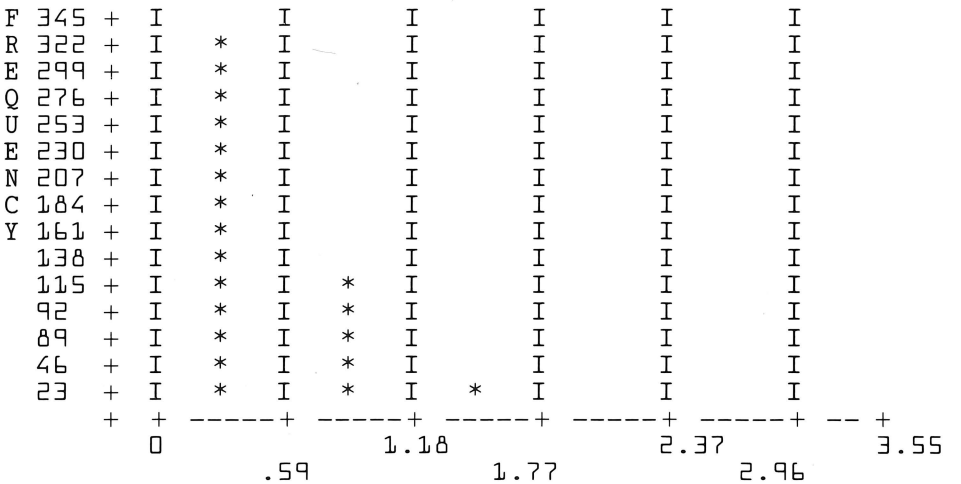


Figure 15.15—Frequency Histogram Exponential Distribution u=.5, σ^2=.25, n=500

$$f(x) = \begin{cases} \dfrac{1}{(k-1)!}\lambda^k(\lambda x)^{k-1}e^{-\lambda x} & x>0 \\ 0 & \text{Otherwise} \end{cases}$$

where k must be an integer, the mean is k/λ, and the variance is k/λ^2. The parameter k is known as a shape parameter. The exponential distribution is a special case of the Erlang distribution when k=1. The Erlang distribution can be generated by summing k-independent and identically distributed exponential random variables.

Figure 15.16 contains a subroutine that will generate Erlang variates. The main program must supply a value for the shape parameter, K1, and a value for the mean of the exponential distribution, XM. The variable RERL contains the sample generated. A frequency histogram of a sample of 500 numbers appears in Figure 15.17.

```
28480  REM  ******************************************************
28490  REM  SUBROUTINE: ERLANG DISTRIBUTION GENERATOR
28500  REM  MAIN PROGRAM SHOULD INITIALLY RANDOMIZE SEED
28510  REM  MAIN PROGRAM MUST SUPPLY VALUES FOR
28520  REM  SHAPE PARAMETER, K1, AND MEAN OF
28530  REM  EXPONENTIAL DISTRIBUTION, XM
28450  REM  RANDOM NUMBER GENERATED IS RERL
28550  IF K1 < 0 THEN  PRINT "INVALID SHAPE PARAMETER": END
28560 RV = 1
28570  FOR I1 = 1 TO K1
28580 RV = RV *  RND (1)
28590  NEXT I1
28600 RERL =  - XM *  LOG (RV)
28610 RETURN
```

Figure 15.16—Erlang Distribution Generator

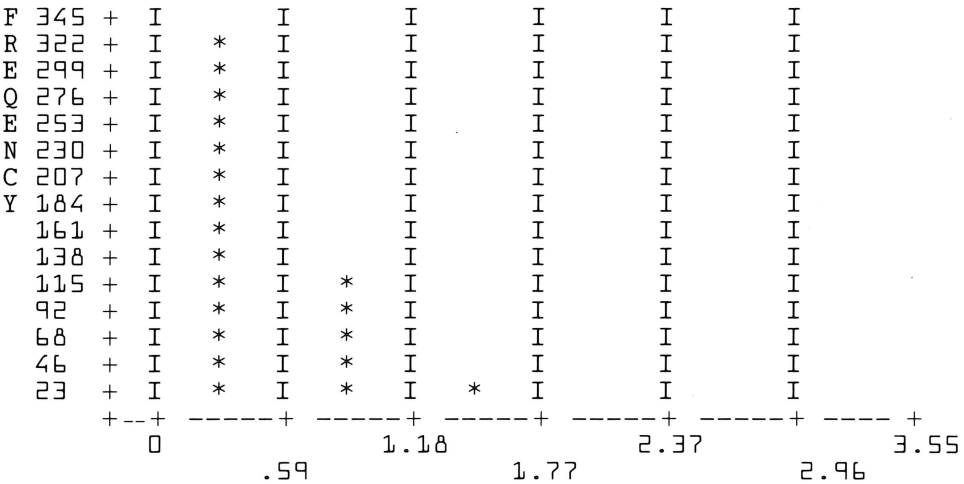


Figure 15.17—Frequency Histogram Erlang Distribution u = .5, σ^2 = .25, N = 500, K = 1

15.5 Next Event Simulation

Computer simulation is often used to analyze a real world physical system. This is done by developing a mathematical-logical model within a computer program that represents the important features of the real system. Once the model is developed, experiments can be performed varying parameters of the model to provide a better understanding of the system. Since models are usually a simplification of the real world, computer simulation is often less expensive and more convenient than manipulation of a real-world system.

Simulation models can be very simple or very complex. An example of a simple simulation model is one that emulates tossing a coin or throwing a die. An example of a complex model is one that simulates the weather. This latter example also illustrates the fact that computer simulation models are simplifications of real-world phenomena. As such, they may not always provide accurate results if the underlying physical relationships are not represented correctly. However, computer simulation is recognized as one of the most powerful techniques we have for analyzing a wide variety of real-world systems. Developing simulation models is an art; consequently, we improve with experience. Some of the programs included in this text can be useful in developing simulation models. Three of these subroutines will be used in the next example presented.

EXAMPLE 3. Develop a program which will simulate a waiting line (queue) with a single server. If the server is busy a new arrival to the system will enter the back of the queue. The queue discipline will be first-in, first-out (FIFO). Assume that the time between arrivals is exponentially distributed with a mean of 5.5 minutes. Also, assume that the service time is normally distributed with a mean of 8 minutes and a standard deviation of 1.5 minutes. The program developed should generate random samples for the arrival and service times (round event times to nearest minute). Because customers do not like to wait in long lines, assume that new arrivals balk (turn away) whenever the queue contains four customers. Therefore, the maximum queue size is four. Use next-event

simulation methodology to simulate this system for 200 minutes. During this time, collect the following statistics:

a) Number of customers served.
b) Average service time.
c) Average waiting time for customers served.
d) Number of customers who balked.

SOLUTION. Figure 15.18 contains a program, called QUEUESIM, that will simulate the system described. Before explaining the program, we will go through the logic used in developing the model.

In this model there are two types of events that occur: arrival of a customer and the completion of service. Consequently, the model must determine each arrival time and each service completion time. These times are determined in the following manner:

a) When an arrival occurs, determine the time interval to the next arrival. In this example the interval is a random time generated from an exponential distribution with a mean of 5.5 minutes between arrivals.
b) When service is initiated, determine the required service time. This will be a random service time generated from a normal distribution having a mean of 8 minutes and a standard deviation of 1.5 minutes.

The program in Figure 15.18 was developed using the following logic:

```
Initialize variables.
Schedule first arrival.
If TIME is less than limit STPSIM,
    Then determine the next event;
    If it is an arrival,
      Then update TIME.
      If number in queue is ≤ 4,
           Then insert arrival in queue.
           Else turn arrival away.
      Determine next arrival time.
      If server is idle
           Then take arrival from queue
           and schedule service completion time
      Else it is a service completion,
           Update TIME and remove
           customer from service
           If number in queue > 0
                Then take a customer from queue
                and schedule service completion time.
    Else stop simulation
         Print simulation results.
```

The QUEUE subroutine (Figure 14.6) from Chapter 14 is used in the above program to store the arrival of customers waiting for service. In addition, the EXPONENTIAL GENERATOR and NORMAL GENERATOR subroutines from this chapter were used to generate random arrival and service times.

```
1000  REM  NEXT EVENT SIMULATION OF A QUEUE WITH A SINGLE SERVER
1010  REM  TIME BETWEEN ARRIVALS IS EXPONENTIALLY DISTRIBUTED WITH MEAN
1020  REM  OF EMU MINUTES. SERVICE TIME IS NORMALLY DISTRIBUTED WITH MEAN
1030  REM  OF NMU MINUTES AND STANDARD DEVIATION OF SD MINUTES.
1040  REM       ARRIVE-NEXT ARRIVAL TIME
1050  REM       SERCOM-NEXT SERVICE COMPLETION TIME
1060  REM
1070 DOS$ =  CHR$ (4): REM   CTRL-D
1080  INPUT "DO YOU WANT TO SPECIFY THE SEED (Y/N) ?";AN$
1090  IF AN$ = "N" THEN  GOTO 1120
1100  PRINT : INPUT "ENTER THE SEED (SEED<0) ? ";SS
1110 XX =  RND (SS)
1120  REM  INITIALIZE VARIABLES
1130 SRV% = 0: REM    NUMBER SERVED
1140 TST% = 0: REM    TOTAL SERVICE TIME
1150 MXQ% = 4: REM    MAXIMUM QUEUE LENGTH
1160 TWTA% = 0: REM   TOTAL WAITING TIME
1170 AWAY% = 0: REM   NUMBER CUSTOMERS TURNED AWAY
1180 TIME% = 0: REM   SYSTEM TIME
1190 IBLE = 0: REM    SERVER IDLE
1200 NOQ% = 0: REM    NUMBER IN QUEUE
1210 EMU = 5.5: REM   MEAN ARRIVAL RATE
1220 NMU = 8: REM     MEAN SERVICE RATE
1230 SD = 1.5: REM    SERVICE STANDARD DEVIATION
1240 QSIZ% = 10: REM  SET QUEUE ARRAY SIZE FOR Q SUB.
1250  PRINT : INPUT "ENTER TIME TO TERMINATE SIMULATION ? ";STPSIM%
1260  GOSUB 24880: REM    INITIALIZE QUEUE ARRAY
1270  GOSUB 1610: REM    SCHEDULE FIRST ARRIVAL
1280  REM  INITIALIZE SERVICE COMPLETION TO LARGE NUMBER SO NEXT
1290  REM  EVENT WILL BE AN ARRIVAL
1300 SERCOMP% = STPSIM% + 1
1310  PRINT : INPUT "TRACE MODEL PROGRESS ON PRINTER (Y/N)? ";Y$
1315  IF Y$ = "Y" THEN  PRINT DOS$;"PR#1"
1320  PRINT : PRINT  TAB( 9)"SIMULATION IN PROGRESS": PRINT
1330  REM  DETERMINE NEXT EVENT
1340  IF ARRIVE% > SERCOMP% THEN 1420
1350  REM  NEXT EVENT IS AN ARRIVAL
1360 TIME% = ARRIVE%
1370  IF TIME% >  = STPSIM% THEN 1480: REM  SIMULATION OVER
1380  GOSUB 1780
1390  IF IBLE <  > 0 THEN 1330: REM  DETERMINE NEXT EVENT
1400  GOSUB 1980
1410  GOTO 1330: REM   DETERMINE NEXT EVENT
1420  REM  NEXT EVENT IS A SERVICE COMPLETION
1430 TIME% = SERCOMP%
1440  IF TIME% >  = STPSIM% THEN 1480: REM   SIMULATION OVER
1450  GOSUB 2140: REM     PROCESS SERVICE COMPLETION
1460  GOTO 1330: REM      DETERMINE NEXT EVENT
1470  REM  ***************************************************************
1480  REM  SIMULATION RESULTS
1490  IF Y$ = "Y" THEN  PRINT DOS$;"PR#1"
1500  PRINT "     NUMBER AVE.TURNED AVE.SERVICE WAIT"
1510  PRINT "TIME SERVED    AWAY       TIME     TIME"
1520  PRINT "======================================="
1530 X = TWTA% / SRV%
1540 X =  INT (X * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
1550 Y = TST% / SRV%
1560 Y =  INT (Y * 10 ^ 2 + .5) /  INT (10 ^ 2 + .5)
1570  PRINT TIME%; TAB( 8);SRV%; TAB( 16);AWAY%; TAB( 27);Y; TAB( 36);X
```

```
1580  PRINT : PRINT  TAB( 11)"END OF SIMULATION"
1590  PRINT DOS$;"PR#0": END
1600  REM  ************************************************************************
1610  REM  SUBROUTINE: GENERATE ARRIVALS
1620 XM = EMU
1630  GOSUB 28410: REM  GENERATE A RANDOM EXPONENTIAL DEVIATE
1640 ARRIVE% = TIME% +  INT (RXP + .5): REM  TIME NEXT ARRIVAL ROUNDED TO INTEGER
1650  IF Y$ <  > "Y" GOTO 1690
1660  PRINT DOS$;"PR#1"
1670  PRINT "NEXT ARRIVAL TIME ";ARRIVE%
1680  PRINT DOS$;"PR#0"
1690  RETURN
1700  REM  ************************************************************************
1710  REM  SUBROUTINE: GENERATE SERVICE COMPLETIONS
1720 XM = NMU
1730  GOSUB 28090: REM  GENERATE A RANDOM NORMAL DEVIATE
1740 WRK% =  INT (RNR + .5): REM  SERVICE TIME ROUNDED TO INTEGER
1750 SERCOMP% = TIME% + WRK%: REM  COMPLETION TIME
1760  RETURN
1770  REM  ************************************************************************
1780  REM  SUBROUTINE: PROCESS NEW ARRIVAL
1790  IF NOQ% = MXQ% THEN 1890: REM  TURN CUSTOMER AWAY
1800 IDTA = TIME%
1810  GOSUB 24940
1820  IF IL <  > 0 THEN  PRINT "ERROR": END : REM  ERROR IN QUEUE SUBROUTINE
1830 NOQ% = NOQ% + 1
1840  IF Y$ <  > "Y" GOTO 1880
1850  PRINT DOS$;"PR#1"
1860  PRINT "NO IN QUEUE AT ARRIVAL ";NOQ%,"TIME ";TIME%
1870  PRINT DOS$;"PR#0"
1880  GOTO 1940
1890 AWAY% = AWAY% + 1
1900  IF Y$ <  > "Y" GOTO 1940
1910  PRINT DOS$;"PR#1"
1920  PRINT "NUMBER TURNED AWAY ";AWAY%,"TIME ";TIME%
1930  PRINT DOS$;"PR#0"
1940  REM  SCHEDULE NEW ARRIVAL
1950  GOSUB 1610: REM    ARRIVAL SUBROUTINE
1960  RETURN
1970  REM  ************************************************************************
1980  REM  SUBROUTINE: INITIATE SERVICE COMPLETION
1990 IBLE = 1: REM  NOT IDLE
2000 SRV% = SRV% + 1: REM  NUMBER SERVED
2010  GOSUB 25050: REM  REMOVE CUSTOMER FROM QUEUE
2020 NOQ% = NOQ% - 1
2030 TWTA% = TWTA% + TIME% - IDTA: REM   TOTAL WAITING TIME
2040  GOSUB 1710: REM   OBTAIN SERVICE INTERVAL
2050 TST% = TST% + WRK%: REM  TOTAL SERVICE TIME
2060  IF Y$ <  > "Y" GOTO 2120
2070  PRINT DOS$;"PR#1"
2080  PRINT "SERVICE COMPLETION TIME ";SERCOMP%,"TIME ";TIME%
2090  PRINT "NO. IN Q ";NOQ%;" , WAIT TIME ";TIME% - IDTA;
2100  PRINT ", NUMBER SERVED ";SRV%
2110  PRINT DOS$;"PR#0"
2120  RETURN
2130  REM  ************************************************************************
2140  REM  SUBROUTINE PROCESS SERVICE COMPLETIONS
2150  IF NOQ% <  > 0 THEN 2240
2160 IBLE = 0: REM   SERVER IS IDLE
2170  REM  MAKE SURE A COMPLETION CANNOT OCCUR BEFORE AN ARRIVAL
2180 SERCOMP% = STPSIM% + 1
2190  IF Y$ <  > "Y" GOTO 2230
2200  PRINT DOS$;"PR#1"
2210  PRINT "QUEUE IS EMPTY:TIME ";TIME%
2220  PRINT DOS$;"PR#0"
2230  RETURN
2240  GOSUB 1980: REM  INITIALIZE SERVICE
2250  RETURN
2260  REM  ************************************************************************
```

```
24800  REM  QUEUE SUBROUTINES
24810  REM  THESE SUBROUTINES WILL INITIALIZE A QUEUE,
24820  REM  AND PERFORM DATA ADDITIONS AND DELETIONS
24830  REM  TO A QUEUE. THE QUEUE IS MAINTAINED IN
24840  REM  THE ARRAY Q. FQ% IS THE FRONT OF QUEUE
24850  REM  POINTER, AND RQ% IS THE REAR OF QUEUE POINTER
24860  REM  ***************************************************************
24870  REM  SUBROUTINE: QUEUE INITIALIZE
24880  DIM Q(QSIZ%)
24890 FQ% = 0: REM  DENOTE EMPTY QUEUE
24900 FQ% = RQ%
24910  RETURN
24920  REM  ***************************************************************
24930  REM  SUBROUTINE: QUEUE ADD
24940 IL = 0: REM  SET NO ERROR CONDITION
24950  IF RQ% = (FQ% - 1) THEN 25010: REM  IS QUEUE FULL?
24960  IF (FQ% = 0) AND (RQ% = QSIZ%) THEN 25010
24970 Q(RQ%) = IDTA: REM  ADD TO QUEUE
24980 RQ% = RQ% + 1
24990  IF RQ% > QSIZ% THEN RQ% = 1
25000  RETURN
25010  PRINT : PRINT "QUEUE FULL"
25020 IL = 1: RETURN : REM  SET ERROR CONDITION
25030  REM  ***************************************************************
25040  REM  SUBROUTINE: QUEUE DELETE
25050 IL = 0: REM  SET NO ERROR CONDITION
25060  IF RQ% = FQ% THEN 25110: REM  IS QUEUE EMPTY?
25070 IDTA = Q(FQ%)
25080 FQ% = FQ% + 1
25090  IF FQ% > QSIZ% THEN FQ% = 0
25100  RETURN
25110  PRINT : PRINT "QUEUE EMPTY"
25120 IL = 1: RETURN
28080  REM  ***************************************************************
28090  REM  SUBROUTINE: NORMAL DISTRIBUTION GENERATOR
28100  REM  MAIN PROGRAM SHOULD INITIALLY RANDOMIZE SEED
28110  REM  MAIN PROGRAM MUST SUPPLY MEAN, XM,
28120  REM  AND STANDARD DEVIATION, SD
28130  REM  RANDOM NUMBER GENERATOR IS RNR
28140  IF NRN = 1 THEN 28240
28150 A1 = 2 *  RND (1) - 1
28160 A2 = 2 *  RND (1) - 1
28170 S = A1 * A1 + A2 * A2
28180  IF S >  = 1 THEN 28150
28190 R1N = A1 *  SQR (( - 2 *  LOG (S)) / S)
28200 R2N = A2 *  SQR (( - 2 *  LOG (S)) / S)
28210 RNR = XM + R1N * SD
28220 NRN = NRN + 1
28230  RETURN
28240 RNR = XM + R2N * SD
28250 NRN = 0
28260  RETURN
28400  REM  ***************************************************************
28410  REM  SUBROUTINE: EXPONENTIAL DISTRIBUTION GENERATOR
28420  REM  MAIN PROGRAM SHOULD INITIALLY RANDOMIZE SEED
28430  REM  MAIN PROGRAM MUST SUPPLY MEAN, XM
28440  REM  RANDOM NUMBER GENERATOR IS RXP
28450 URN =  RND (1)
28460 RXP =  - XM *  LOG (URN)
28470  RETURN
```

Figure 15.18—Next Event Simulation Program

The technique of next-event simulation is illustrated in the example. Although the program will simulate 200 minutes of activity (STPSIM = 200), the system time (TIME) does not advance by seconds or minutes. Instead, TIME is reset each time an event occurs

(a customer arrival or a service completion). Since only two events can occur, it is not necessary to look at the time between the occurrence of one of these events because nothing can happen. With practice, next-event simulation models are relatively easy to develop and they execute faster than a "clock oriented" model in which time is incremented by fixed intervals.

The program in Figure 15.18 contains PRINT statements that will permit you to trace the progress of a simulation run. You are given the option of skipping these print statements. Also, you may easily change the parameters of the model, such as maximum queue number (MXQ), simulation run length (STPSIM), mean arrival rate (EMU), and mean service rate and standard deviation (NMU and SD). Varying these parameters lets you study the system being modeled. The final results from running this model for 200 minutes with a random number seed of -1 may be seen in Figure 15.19.

Time	Number Served	Turned Away	Ave. Service Time	Wait Time
200	22	5	7.55	20.68

Figure 15.19—Simulation Results

15.6 Summary

At the beginning of this chapter, we explained how to use the BASIC numeric function RND. We then presented a subroutine (named ORGRND) which would generate random numbers using the congruential method. A chi-square goodness of fit subroutine was developed to determine if the numbers generated by the RND and ORGRND subroutines were not random. Both subroutines passed the chi-square test. Next, random number generators (subroutines) were provided for the uniform, normal, Poisson, exponential, and Erlang distributions.

In the last section of this chapter, we presented the next-event simulation technique. This is one of the most useful computer techniques available. In fact, simulation is so useful that several general purpose simulation languages have been developed for minicomputers and mainframe computers. These languages reduce the programming effort required to develop a model and to generate statistical results. One popular language is SLAM, which is well explained by Pritsker (1979). To date, most of these languages have been developed using the FORTRAN programming language. However, some simulation languages are being developed especially for microcomputers. Consequently, it will be feasible to develop some simulation models on a microcomputer.

References

Marsaglia, G., and T. A. Bray, "A Convenient Method for Generating Normal Variables," *SIAM Review*, Vol. 6, No. 3, pp. 260–64, 1964.

Pritsker, A. Alan B. and Claude Dennis Pegden, *Introduction to Simulation and SLAM*, Halsted Press, New York, 1979.

Shannon, Robert E., *Systems Simulation the Art and Science*, Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

Exercises

For exercises 1–5, you can use the subroutines in Chapter 2 to calculate the mean and variance of the numbers generated and to plot a histogram.

1. Write a program that will simulate the tossing of a coin 20 times. Keep track of the number of heads and tails that occur. Do you think that your program realistically simulates a coin tossing event?
2. Generate 50 uniformly distributed random numbers in the interval of $50 \leq x \leq 99$. Use the chi-square goodness of fit test to determine if the resulting numbers are not uniformly distributed.
3. Generate 300 normally distributed numbers having a mean of 15 and variance of 3.
4. Using the Poisson distribution, generate 200 numbers that have a mean of 20.
5. Generate 400 numbers using the Erlang Distribution Generator subroutine. Set the shape parameter equal to 3 and use a exponential distribution mean of 2.
6. Simulate the process of a blind man crossing a street. While he walks across the street several events can occur. These events and the associated probabilities of occurring are:

   a. Walks across the street without falling down; probability of .67.
   b. Falls down while walking across the street; probability of .33.
   c. Hit by a truck while walking in street; probability of .15.
   d. Hit by a truck while lying in the street; probability of .33.

   Simulate the man crossing the street 30 times keeping track of how many times he successfully crosses the street. What is the probability of the blind man successfully crossing the street?

7. Using the next-event simulation model of a single queue, single server system provided in this chapter, simulate 500 minutes of activity. The average time between arrivals is to be 9 and the average service time is to be 9 with a variance of 4.
8. Develop a next-event simulation model of a single queue with two servers. Run this model for 300 time periods. The time between arrivals is to be exponentially distributed with a mean of 8. The service time for each server is to be normally distributed with a mean of 4 and a variance of 2.

Answers to Exercises

Answers to selected exercises are provided below. A few exercises have been left to the reader to answer.

Chapter 1

1. A source program is a program written in a higher language such as BASIC that must be translated to machine language before it can be executed by a computer.

   An object program is a program in machine language that can be executed without additional translation.
2. A BASIC interpreter is a program that translates a source (BASIC) statement into machine language statements. These machine language statements are then executed before another source statement is translated.

   A BASIC compiler translates the entire source program into a machine language program (object program). After which, the machine language program is executed.
3. A compiler should be used when program execution time is to be minimized.

   An interpreter should be used when memory is limited and when ease of program debugging is important.
4. Significant features of the Apple Computer:
   1. Typewriter-like keyboard.
   2. Amount of software available.
   3. Expandable memory.
   4. Flexibility of application.
   5. High resolution monitor available.
   6. Supports a color monitor.
   7. High resolution graphics.

5.
```
PRINT CHR$(4);"RUN EXSMOOTH"
```
6.
```
&H
LOAD QUEUE
&M
```
7.
```
 5 PRINT
10 N%=40
20 X=3/10
30 FOR I=1 TO N%
40 X=X/10
50 PRINT I,SIN(X),X
60 NEXT I
70 END
```

I	SIN(X)	X
1	.0299955002	.03
2	2.9999955E-03	3E-03
3	2.99999995E-04	3E-04
4	2.99999982E-05	3E-05
5	2.99999925E-06	3E-06
6	2.9999821E-07	3E-07
7	2.99983924E-08	3E-08
8	2.99726771E-09	3E-09
9	0	3E-10
10	0	3E-11
11	0	3E-12

12	0	3E-13
13	0	3E-14
14	0	3E-15
15	0	3E-16
16	0	3E-17
17	0	3E-18
18	0	3E-19
19	0	3E-20
20	0	3E-21
21	0	3E-22
22	0	3E-23
23	0	3E-24
24	0	3E-25
25	0	3E-26
26	0	3E-27
27	0	3E-28
28	0	3E-29
29	0	3E-30
30	0	3.00000001E-31
31	0	3.00000001E-32
32	0	3.00000001E-33
33	0	3.00000001E-34
34	0	3.00000001E-35
35	0	3.00000001E-36
36	0	3.00000001E-37
37	0	3.00000001E-38
38	0	3.00000001E-39
39	0	0
40	0	0

8.

A% = 2	A% = 2
B = 3	B = 3
X = .6666667	X% = 1
A = 2	A = 2
B% = 3	B% = 3
X = .6666667	X% = 1
A% = 2	A% = 2
B% = 3	B% = 3
X = .6666667	X% = 1

9. Using the program below, an incorrect answer was obtained when the difference of X and X1 was equal to 2 (at I = 8) and 0 (at I = 9,10).

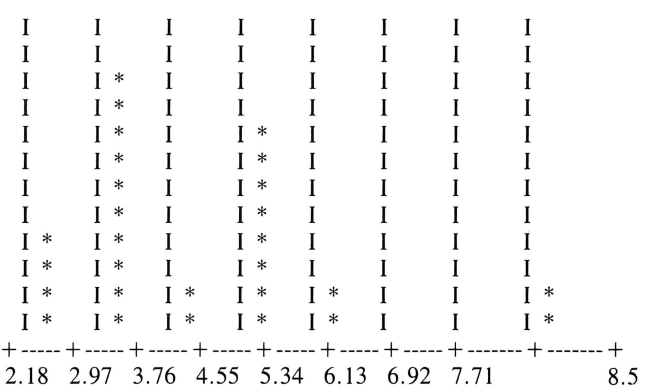
```
20 X=50
30 N%=10
40 FOR I=1 TO N%
50 X1=X*10
60 X=X1+1
70 XD=X-X1
80 PRINT I,X,X1,XD
90 NEXT I
100 END
```

I	X	X1	Difference
1	501	500	1
2	5011	5010	1
3	50111	50110	1

4	501111	501110	1
5	5011111	5011110	1
6	50111111	50111110	1
7	501111111	501111110	1
8	5.01111111E+09	5.01111111E+09	2
9	5.01111111E+10	5.01111111E+10	0
10	5.01111111E+11	5.01111111E+11	0

Chapter 2

1. Mode: 3.2 and 4.6 each occurred twice
   Median: 3.2
   Mean: 3.25238095
   Range: 7.9
   Variance: 3.86725624
   Standard Deviation: 1.96653407

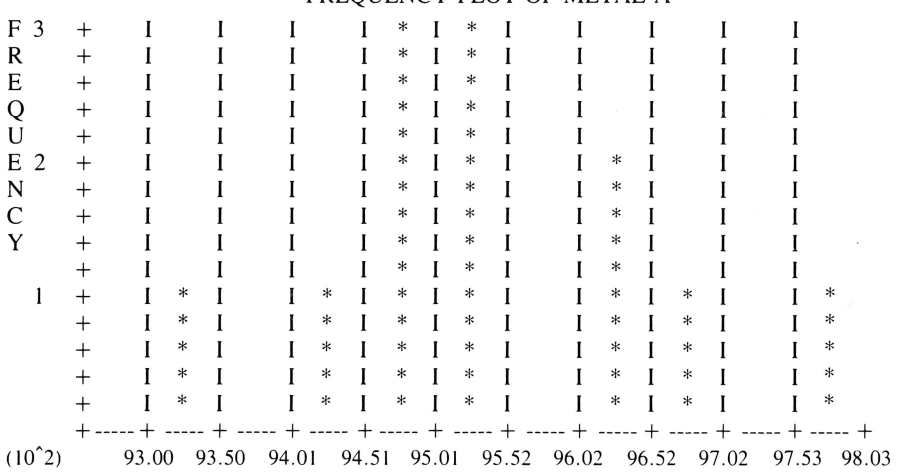


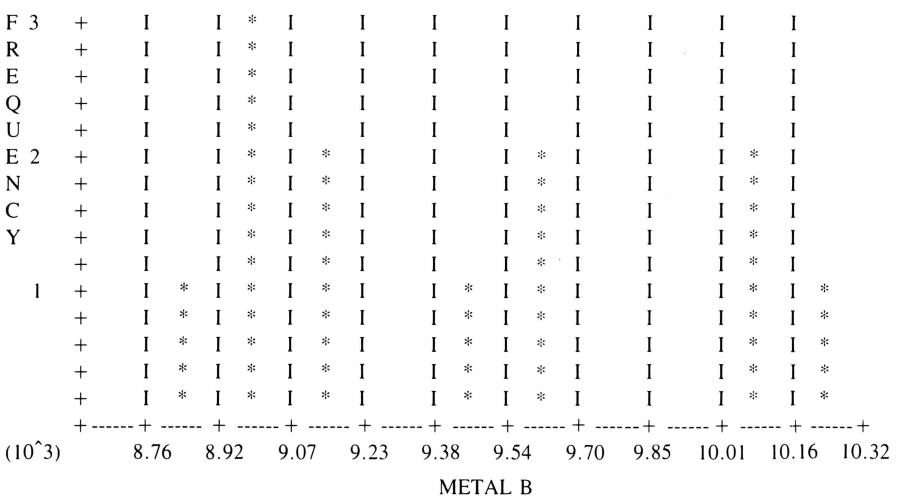
2. Metal A

   Mode: No value occurred more than once
   Median: 9511.5
   Mean: 9542.91667
   Range: 503
   Variance: 16155.4063
   Standard Deviation: 127.103919

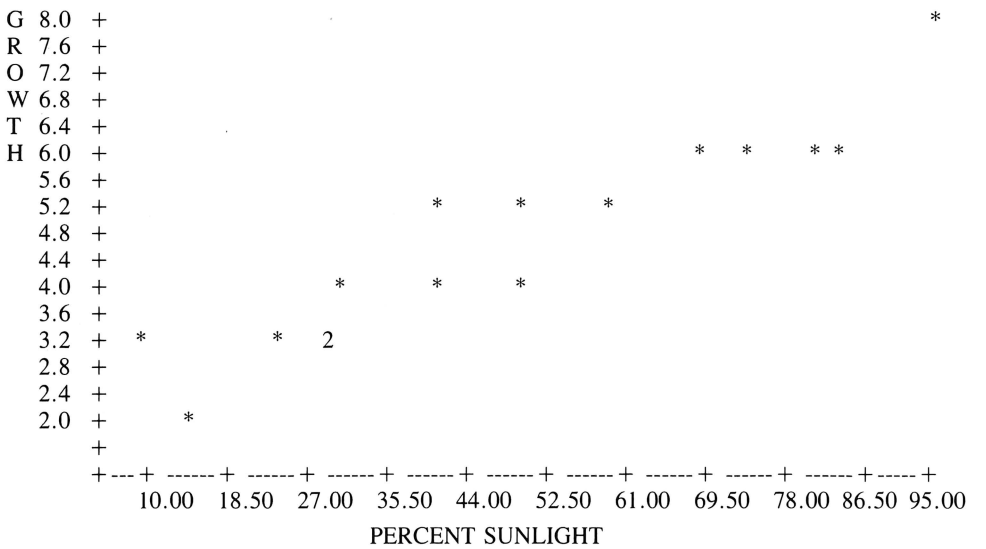
   Metal B

   Mode: No value occurred more than once
   Median: 9358
   Mean: 9439.66667
   Range: 1560
   Variance: 240796.073
   Standard Deviation: 490.709765


FREQUENCY PLOT OF METAL A
FREQUENCY
(10^2)
93.00 93.50 94.01 94.51 95.01 95.52 96.02 96.52 97.02 97.53 98.03
METAL A


FREQUENCY PLOT OF METAL B
FREQUENCY
(10^3)
8.76 8.92 9.07 9.23 9.38 9.54 9.70 9.85 10.01 10.16 10.32
METAL B

3. See plot: Sunlight does not appear to be the only factor relevant to plant growth.



Chapter 3

1.

$$\underline{A} \times \underline{B} = \begin{bmatrix} 32 & 15 & 37 \\ 64 & 30 & 14 \\ 24 & 13 & 67 \end{bmatrix} \qquad \underline{B} \times \underline{A} = \begin{bmatrix} 52 & 54 & 20 \\ 6 & 54 & 18 \\ 37 & 61 & 23 \end{bmatrix}$$

For $A \times B = B \times A$, let

$$A = \begin{bmatrix} 3 & 5 & 1 \\ 6 & 0 & 2 \\ 1 & 9 & 3 \end{bmatrix} \qquad B = A^{-1} = \begin{bmatrix} 0.225 & 0.075 & -0.125 \\ 0.2 & -0.1 & 0 \\ -0.675 & 0.275 & 0.375 \end{bmatrix}$$

3. a. let $x_1 = 5$, $x_2 = 6$, $x_3 = 3$
   then $b_1 = 72$, $b_2 = 35$, $b_3 = 3$
   b. let $x_1 = 0.5$, $x_2 = 0.25$, $x_3 = -2$
   the $b_1 = -11$, $b_2 = .5$, $b_3 = 2.25$

4. Determinant of the resulting matrix is zero.

7. $| k\underline{A} | = k^2 | \underline{A} |$

8.

$$A^{-1} = \begin{bmatrix} 0 & 1 & -2 & -3.8805E\text{-}11 \\ .0909090908 & -.727272728 & 3.5454544 & -.272727273 \\ .12121212 & -.636363636 & 1.39393939 & -.0303030303 \\ -.0909090909 & -.272727272 & -.545454546 & .272727273 \end{bmatrix}$$

Chapter 4

1. Participate = 9.704 + 1.551 Reagent
   Coefficient of determination = .734
2. $e^{\text{Distance}} = 3.087 + .309$ MPH, or
   Distance = $21.922e^{.039\ \text{MPH}}$
   Coefficient of determination = .995
   Predicted distance for 60 MPH = 227.140 feet
3. Salary = 14786.3 + 1912.3 Years
   Coefficient of determination = .970
4. Yield = −11.54 + 15.54 *LOG(Lots)
   Coefficient of determination = .972
   Predicted yield for lot number 400 is .816
5. Viscosity = 5.561 − .0089 Temperature
   $e^{\text{Viscosity}} = 260.19e^{-.0089\ \text{Temperature}}$
   Coefficient of determination = .994
6. Y = −140.14 + 5.05 x1 + .67 x2
   Coefficient of determination = .838
   Note that you will not get this answer if you use the stepwise procedure. This is an example of when multiple regression will give better results.
7. Y = 14.982 + 3.796 x3
   Coefficient of determination = .9896
   Only x3 is needed in the regression equation.
8. Price = −1.075 + 50.734 Earnings
   Coefficient of determination = .959
   Predicted end-of-year stock price is $41.03

Chapter 5

2. a. No solution
   b. Infinite number of solutions
   c. No solution
   d. Infinite number of solutions
4. Input A = 2.1 ounces
   Input B = 1.25 ounces
   Input C = 4.45 ounces

Chapter 6

1. Stationary point: −.891767197
2. (Some roots are listed below)
   a. x = .69669054, x = 1.13789919
   b. x = 0, x = .951367333, x = −1.28470069
   c. x = −4.35981145, x = 3.34632891

5.

	Maximum number of real roots	Roots you may find: Real	Roots you may find: Complex
a.	1	1.08846817	.731197264 ± .610849441i
			−.538959115 ± .871867562i
b.	0	.800296169	
		1.36841247	−.523950586 ± 1.74509399i
		−.730118257	
		−3.3904455	
c.	1	0	.527036317 ± .49384745i
			−.505306809 ± .494785823i
d.	0	.63781864	.352521148 ± .610609095i
		−.704980575	−.318716727 ± .552469651i

Chapter 7

1. Trapezoidal Rule for h = 0.01 — Legendre-Gauss: 2 subintervals

   a. 130.0016 — a. 130
   b. 2697.3323 — b. 2696.39523

2. Methods presented automatically handle this; no revision required.
3.

Legendre-Gauss	Trapezoidal with 100 subintervals
a. 0.92474705	0.977186056
b. 0.229729363	0.229744058
c. 0.459221848	0.459205651

5. Results found using Legendre-Gauss quadrature

   a. 0.531248966
   b. 0.506867955
   c. 14.8107275

Chapter 8

1. Amount of substance at t = 6 is 9.26744 lbs.
2.

x	0.0	1.0	2.0
y = g(x)	0	0.15524	−0.48108

3. Standard form: $y' = \dfrac{1 - xy - y}{x}$

x	1.0	1.5	2.0
y = g(x)	0	0.25969	0.31439

4.

	Order	Could We Solve
a.	2	No
b.	1	Yes
c.	2	No
d.	1	Yes
e.	1	Yes

5. Expression: $\dfrac{dQ}{dt} = kQ$ where k is a constant

   (in this case, k = −0.02321)

   Amount of Dittmannium left after 24 days = Q (24) = 171.8707

Chapter 9

1. Z = 34.72
   $x_1 = 3.2$, $x_2 = 2.36$, $x_3 = 0.88$
2. Let x_i = number of barrels of fuel type i produced, where
   i = 1:leaded
   i = 2:unleaded
   i = 3:diesel

   Maximize $Z = 0.12x_1 + 0.09x_2 + 0.10x_3$
   s.t.

$$0.4x_1 + 0.5x_2 + 0.7x_3 \leq 400$$
$$0.6x_1 + 0.5x_2 + 0.3x_3 \leq 300$$
$$x_1, x_2, x_3 \geq 0$$

Solution:

$Z = 7600, \quad x_1 = 300 \quad x_2 = 0, \quad x_3 = 400$

3. Let the variables be named so that the letters represent the first letter for each fertilizer/crop combination. That is,

AC = fertilizer A on corn
CS = fertilizer C on soybeans, etc.
unit = bushels

$$\text{Maximize } Z = 2(AC+BC+CC)+1.5(AW+BW+CW)+1.8(AS+BS+CS)$$
$$-(1.333AC+1.389BC+1.667CC)-(0.645AW+BW+0.571CW)$$
$$-(AS+1.563BS+0.667CS)$$
$$= 0.667AC+0.611BC+0.333CC+0.855AW+0.5BW$$
$$+0.929CW+0.8AS+0.237BS+1.133CS$$

s.t.

$$15AC+31AW+20AS \leq 100$$
$$18BC+25BW+16BS \leq 125$$
$$12CC+35CW+30CS \leq 80$$
$$\text{all variables} \geq 0$$

Solution:

$AC = 6.7$ all other variables = 0
$BC = 6.94$
$CS = 2.67$
$Z = 11.17$

4. Let x_1 = number of units of power tool A produced per week
x_2 = number of units of power tool B produced per week

Maximize $Z = 2x_1 + x_2$

s.t.

$$30x_1 + 40x_2 \leq 2400$$
$$45x_1 + 30x_2 \leq 2400$$
$$20x_1 + 25x_2 \leq 2400$$
$$x_1, \quad x_2 \geq 0$$

Solution:

$Z = 106.67$ (although its actual value is irrelevant)
$x_1 = 53.333$
$x_2 = 0$

5. Solution: Produced nothing!
(Can you examine the formulation and tell why?)

Chapter 10

1. a.

Month	Moving Average
1	—
2	—
3	3.43
4	3.50
5	3.70
6	4.23
7	4.70
8	5.10
9	5.43

Estimate for the10th month is 5.43.

b. Using the average of the first three months as the initial forecast value, then

$$F_2 = F_1 + .3(X_1 - F_1)$$
$$= 3.43 + .3(3.5 - 3.43)$$
$$= 3.451$$

The following values were obtained in a similar manner:

Month	Single Exponential Smoothed Forecast
2	3.451
3	3.496
4	3.407
5	3.495
6	3.71
7	4.03
8	4.35
9	4.67
10	5.01

2. For: season length = 6
initialization periods = 18
lead time = 1
alpha = .3
beta = .1
gamma = .1

Period	Forecast
19	31.52
20	31.50
21	33.63
22	35.88
23	38.25
24	40.60

For the initialization phase, the mean squared error was 2.198 and the mean absolute deviation was 1.167.

3. a.

Period	3-Month Moving Averag
1	—
2	—
3	3
4	5
6	9
7	11
8	12
9	15
10	17

b. Using an initial forecast value of $\frac{1+3+5}{3} = 3$, and an alpha = .5, the following results were obtained:

Period	Single Smoothed Value
1	—
2	2
3	2.5
4	3.75
5	5.31
6	7.19
7	9.09
8	11.05
9	13.02
10	15.01

c. Using an initial forecast value of 1 and an initial trend value of 2, the following results were obtained:

Period	Holt's Smoothed Values Alpha = .1 Beta = .1
2	3
3	5
4	7
5	9
6	11
7	13
8	15
9	17
10	19

d. For: season length = 6
initialization periods = 9
lead time = 1
alpha = .1
beta = .1
gamma = .1

Period	Winter's Smoothed Values
2	3
3	5
4	7
5	9
6	11
7	13
8	15
9	17
10	19

Holt's and Winter's methods are the best because these methods were developed to model a process containing a trend.

4. a. Using an initial smoothed value of $\frac{1+4+7+10}{4} = 5.5$

Period	Single Smoothed Value
1	—
2	3.12
3	3.56
4	5.28
5	7.64
6	4.32
7	4.16

8	5.58
9	7.79
10	4.39
11	4.20
12	5.60
13	7.80
14	4.40
15	4.20
16	5.60
17	7.80

b. For: season length = 4
initialization periods = 12
lead time = 1
alpha = .1
beta = .1
gamma = .1

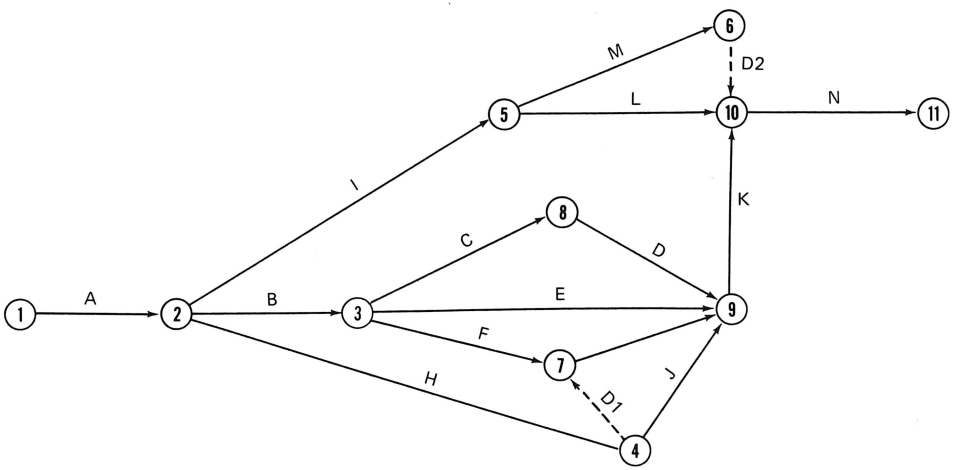
Period	Winter's Smoothed Value
1	—
2	4
3	7
4	10
5	1
6	4
7	7
8	10
9	1
10	4
11	7
12	10
13	4
14	4
15	7
16	10
17	1

5. For: season length = 6
initialization periods = 18
lead time = 1
alpha = .3
beta = .1
gamma = .1

Period	Winter's Smoothed Value
19	26.60
20	29.43
21	34.86
22	35.70
23	42.57
24	49.23
25	40.97
26	42.23
27	47.98
28	47.36
29	52.11
30	58.30

Chapter 11

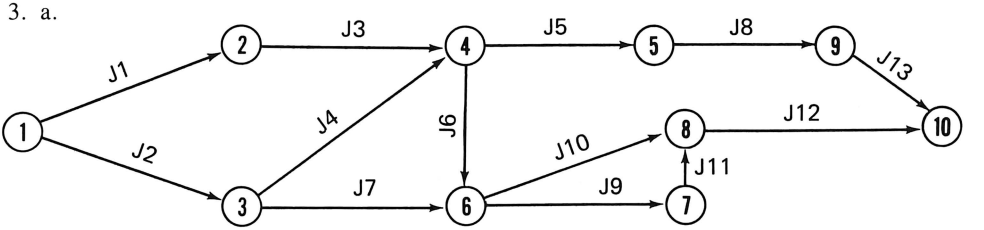
1. a.



b. Critical path: A-I-M-D2-N
Expected project completion date is 19 days.

2. Critical path: B-D-I-J-M-P-O-Q

3. a.



b.

Job	Early Start	Late Start
J1	0	0
J2	0	5
J3	3	3
J4	1	6
J5	8	9
J6	8	8
J7	1	9
J8	12	13
J9	11	12
J10	11	11
J11	12	13
J12	14	14
J13	14	15

c. Critical path: J1-J3-J6-J10-J12
Project duration of 12 days.

4. Critical path: A-C-I-N-K-L-Q-S-R-Y-Z-DC5-END
Project duration of 36 weeks.

Chapter 12

1. Change line 23090 to

   23090 IF X(J + 1)<=X(J) GOTO 23120

2. The following lines should be:

   23710 IF (PV<X(R%)) THEN 23780
   23750 IF (PV>X(L%)) THEN 23810

3. Value for time was 130 seconds. This value will vary with the list being sorted.
4. Value for time was 20 seconds.
5. The bubble sort algorithm required 1 second and the quicksort algorithm required 18 seconds. The bubble sort was much faster because this algorithm can determine when a list is in sorted order, and at that time stop the sorting process.

Chapter 13

1. RAM and ROM memory are primary storage devices. Some secondary storage devices are disk and tape devices.
2. Each track on a floppy disk is divided into sectors. The Apple II has 16 sectors/track and 35 tracks on one side of a floppy disk. Each sector will hold 256 bytes.
3. The primary function of the DOS diskette directory is to maintain a table containing the names of files that have been saved on the diskette.
4. The sequential access method means that data are written and read in a sequential manner starting from the beginning of the storage device. The random access method means that data records can be written to or read from a storage device in a random order.
5. Sequential access input/output modes:

   a. Input
   b. Output
   c. Append

6. a. A record is a group of logically related information.
   b. A buffer is where random access records are temporarily stored before being written to a disk file or where these records are stored after being read from a disk file.
   c. A file is a collection of related information that is usually kept on a disk or tape device.

7,8, and 10.

We have left the writing of these programs up to the reader. The programs should be very similar to the SEQMAN program in this chapter.

9 and 11.

We have left the writing of these programs up to the reader. The programs should be very similar to the RANMAN program in this chapter.

Chapter 14

1. A stack is a data structure in which data are inserted and deleted in a last-in, first-out manner.
2. A queue is a data structure in which data are inserted and deleted in a first-in, first-out manner.
3.
```
100 REM PROBLEM 3, CHAPTER 14
110 DIM A(50)
120 SIZ%=50:REM   SET ARRAY SIZE
130 GOSUB 24560:REM   INITIALIZE STACK
140 REM PLACE AVAILABLE SPACE IN ARRAY ON STACK-LIFO
150 FOR I=1 TO 50
160 IDTA = I
170 GOSUB 24620:REM   PUSH I ON STACK
180 IF IL<>0 THEN 410
190 NEXT I
200 REM PLACE FIRST DATA ELEMENT IN ARRAY A
```

```
210 INPUT "ENTER DATA ELEMENT"; VLUE
220 GOSUB 24710:REM  POP STACK
230 IF IL<>THEN 410
240 A(IDTA)=VLUE
250 INPUT "CONTINUE(Y/N)"; Y$
260 IF Y$="Y" THEN 210
270 STOP
410 PRINT "ERROR"
420 END
```

4. A queue data structure will be utilized using the subroutines from this chapter. If 25 readings are to be maintained, then the queue size must be set to 26 (one element cannot be used—see explanation of queue subroutines).

```
110 QSIZ%=26:REM  SET ARRAY SIZE
120 GOSUB 24870:REM  INITIALIZE QUEUE
130 REM DATA CAN NOW BE PLACED IN THE QUEUE
140 REM WHEN THE QUEUE BECOMES FULL, THE DATA ELEMENT AT
150 REM FRONT OF QUEUE SHOULD BE REMOVED.
160 REM THEN A NEW DATA ELEMENT CAN BE
170 REM ADDED TO THE REAR OF THE QUEUE.
180 REM THIS CAN BE DONE AS FOLLOWS
190 GOSUB 25040:REM  TAKE DATA FROM QUEUE
200 IF IL<>0 THEN 410
210 REM PLACE NEW ELEMENT IN QUEUE
220 IDTA=VALUE
230 GOSUB 24930:REM  PLACE DATA IN QUEUE
    .
    .
    .
    .
410 PRINT "ERROR"
420 END
```

5. The writing of this program is left to the reader. The bidirectional linked list subroutines from this chapter can be used. Since the data consists of string and numeric type data, it will be easier to maintain if the numerical values are converted to strings before they are placed into the linked lists. Since there are four inspectors, four linked lists are required.

Chapter 15

1. There are several ways to write this program. We will generate a random number X between 0 and 1. If $0 \leq X < .5$ we will say the toss of the coin resulted in a head; otherwise, the result was a tail.

```
 10 INPUT "INITIAL VALUE FOR RND ARGUMENT";F
 15 A=RND(F):REM  ASSUME INITIAL ARGUMENT IS NEGATIVE
 20 INPUT "HOW MANY COIN TOSSES ARE TO BE SIMULATED?; N
 30 FOR I=1 TO N
 40 X = RND(1)
 50 IF X<.5 THEN HEAD=HEAD+1
 60 IF X>=.5 THEN TAIL=TAIL+1
 70 NEXT I
 80 PRINT "HEADS="; HEAD
 90 PRINT "TAILS="; TAIL
100 END
```

2.
```
 10 DIM AY(50)
 20 INPUT "INITIAL VALUE FOR RND ARGUMENT";F
 25 G=RND(F):REM  ASSUME INITIAL ARGUMENT IS NEGATIVE
 30 A=50
 40 B=99
 50 INPUT "HOW MANY NUMBERS ARE TO BE GENERATED";N
 60 FOR I=1 TO N
 70 GOSUB 2800: REM  UNIFORM DISTRIBUTION
 80 AY(I)=RUFM
 90 NEXT I
100 GOSUB 27200:REM  CHI-SQUARE FIT TEST
110 END
```

```
 10 DIM X(300)
 20 INPUT "INITIAL VALUE FOR RND ARGUMENT";F
 25 A=RND(F):REM  ASSUME INITIAL ARGUMENT IS NEGATIVE
 30 XM=15
 40 SD=SQR(3)
 50 INPUT "HOW MANY NUMBERS ARE TO BE GENERATED";N
 60 FOR I=1 TO N
 70 GOSUB 28080:REM  NORMAL DISTRIBUTION
 80 X(I)=RNOR
 90 NEXT I
100 END
```

4.
```
 10 DIM X(200)
 20 INPUT "INITIAL VALUE FOR RND ARGUMENT";F
 25 A=RND(F):REM  ASSUME INITIAL ARGUMENT IS NEGATIVE
 30 XM=20
 40 INPUT "HOW MANY NUMBERS ARE TO BE GENERATED";N
 50 FOR I=1 TO N
 60 GOSUB 28270:REM  POISSON DISTRIBUTION
 70 X(I)=RPOI
 80 NEXT I
 90 END
```

5.
```
 10 DIM X(400)
 20 INPUT "INITIAL VALUE FOR RND ARGUMENT";F
 25 A=RND(F):REM  ASSUME INITIAL ARGUMENT IS NEGATIVE
 30 K1=3
 40 XM=2
 50 INPUT "HOW MANY NUMBERS ARE TO BE GENERATED";N
 60 FOR I=1 TO N
 70 GOSUB 28480:REM  ERLANG DISTRIBUTION
 80 X(I)=RERL
 90 NEXT I
100 END
```

6. There are several ways to answer this problem.

```
 10 INPUT "INITIAL VALUE FOR RND ARGUMENT";F
 15 A=RND(F):REM  ASSUME INITIAL ARGUMENT IS NEGATIVE
 20 FOR I=1 TO 30
 25 REM DID HE FALL?
 30 X=RND(1)
 40 IF X<=.66 THEN 100:REM  DID NOT FALL
 50 REM FELL DOWN
 55 REM WAS HIT LYING IN THE STREET?
 60 X=RND(1)
```

```
 70 IF X>.33 THEN 100:REM   WAS NOT HIT AFTER FALLING
 80 REM HIT WHILE LYING IN STREET
 90 HIT=HIT+1:GOTO 130
 95 REM WAS HE HIT WALKING?
100 X=RND(1)
110 IF X<=.15 THEN HIT=HIT+1
120 IF X>.15 THEN SUCCESS=SUCCESS+1
130 NEXT I
140 PRINT "PROBABILITY OF BEING HIT=";HIT/30
150 PRINT "PROBABILITY OF SUCCESSFULLY CROSSING";SUCCESS/30
160 END
```

7. The following changes must be made:

```
1180 EMU= 9
1190 NMU=8
1200 SD=2
```

Index

—A—

Addition, matrices, 38–40
Apple computers,
  applications of, 3–4
  features of, 1–2
Arrays, 269–270

—B—

BASIC interpreter, 4–5
Bubble sort, 233–234
  program for, 234

—C—

Chi-square goodness of fit test, 290–294
  subroutine for, 292
Critical Path Method (CPM), 213–232
  problem example, 222–229
  program for, 215–222
  technique, 213–215
Curve fitting, linear regression and, 51–106

—D—

Data analysis, 18–32
  program for, 19–21
Data plots, 17–18
Data reduction, 11–34
  basics of, 11–12
  data analysis, 18–32
  data plots, 17–18
  measures of central tendency, 12–15
  measures of dispersion, 15–17
  plotting subroutines, 18–32
  see also Statistics
Data structures, 269–286
  arrays, 269–270
  linked lists, 274–284
    bidirectional, 280–284
    linked queues, 277–280
    subroutine for, 278–279
  queues, 272–274
    structure, 273
  stacks, 270–272
    subroutine for, 271
Determinant of a matrix, 43–46
Differential equations,
  numerical solutions to, 147–156
    overview of, 145–146
    Runge-Kutta method, 147–154
Disk data files, 243–268
Disk organization, 244
Disks, recording data on, 243–245
DOS commands, 5–7

—E—

Erlang distribution, 298–299
  generator, 298–299
Exponential distribution, 297–298
  generator, 297–298
Exponential smoothing and forecasting, 181–212
  seasonal, 183–185
  single, 181–183
  trend, 183
Exponential smoothing program, 186–209
Exponentiation, matrix, 42–43

—F—

File management,
  random, program for, 258–267
  sequential, program for, 250–254
Files,
  disk data, 243–268
  random access, 254–267
    reading from, 257–258
    writing to, 255–257
  sequential access, 245–254
    reading and writing, 246–250
Floppy disks *see* Disks
Forecasting with exponential smoothing, 181–212
Frequency plots, main program for, 30–31

—G—

Gauss-Jordan method,
  solving simultaneous linear equations, 109–114
Gauss-Seidel method,
  solving simultaneous linear equations, 114–119
Gaussian elimination, 43–44

—I—

Inverse of a matrix, 46–49

—L—

Legendre-Gauss quadrature, numerical integration and, 138–143
Linear algebra, linear programming and, 163–165
Linear equations, simultaneous,
  solution of, 107–122
    Gauss-Jordan method, 109–114
    Gauss-Seidel method, 114–119
    matrix inverse method, 108–109, 110
Linear programming, 157–180
  basic formulation, 158–163
    example, 161–163
    graphical solution, 159–161
  solution methods, 163–166
    linear algebra, 163–165
    simplex procedure, 165–177
Linear regression,
  coefficient correlation, 53
  coefficient of determination, 53
  curve fitting and, 51–106
  product rejection example, 57
  program, 55
  see also Multiple regression
Linked lists, 274–284
  bidirectional, 280–283
    subroutines, 282–283
    using, 285
  linked queues, 277–280
  subroutines, 278–279

—M—

Matrices, 35–48
  defined, 35–36
  dimensions subroutine, 36
  operations, 36–49
    addition and subtraction, 38–40
    determinant, 43–46
    exponentiation, 42–43
    inverse, 46–49
    multiplication, 40–42
    multiplication by a scalar, 40
    transpose, 36–38
Matrix inverse method,
  solving simultaneous linear equations, 108–109, 110
Matrix multiplication subroutine, 41
Multiple regression, 57–58
  stepwise, 58–104
    program for, 59–70
    use of program, 72–104
  see also linear regression
Multiplication
  by a scalar, 40
  matrices, 40–42

—N—

Newton-Raphson method,
  complex numbers, 132–134
  roots of polynomials and, 124–135
Normal distribution, 295–296
  generator, 296
Numerical integration, 137–144
  basics of, 137
  numerical methods and, 138–143
    Legendre-Gauss quadrature, 141–143
    trapezoidal rule, 138–141

—P—

Poisson distribution, 296–297
  generator, 297
Polynomials, roots of, 123–136
  complex, 130–135
  Newton-Raphson method, 124–135
    multiple starting points and roots, 126
    procedure of, 125
    subroutine for, 128–130
    unfortunate iterative point, 127
Precision, 7
  loss of, 7–9
Programs,
  bubble sort, 234
  CMP, 215–222
  data analysis, 19–21
  exponential smoothing, 186–209
  Legendre-Gauss quadrature, 142
  linear regression, 55
  matrix inverse, 48
  next event simulation, 301–303
  plotting joint observations, 31–32
  quicksort, 241
  random file management, 258–267
  random numbers generating, 291
  Runge-Kutta method, 150
  sequential file management, 250–254
  stepwise multiple regression, 59–70
  trapezoidal rule, 140
  variable subintervals, 141
  see also Subroutines

—Q—

Queues, 272–274
  structure, 273
Quicksort, 235–239
  program, 241

—R—

Random access files, 254–267
  reading from, 257–258
  writing to, 255–257
Random file management program, 258–267

Random number generators,
development of, 289–290
specific distributions, 294–299
Erlang distribution, 298–299
exponential distribution, 297–298
normal distribution, 295–296
Poisson distribution, 296–297
uniform distribution, 294–295
Random numbers, 287–299
chi-square goodness of fit test, 290–294
subroutine for, 292
function, 287–289
generating program, 291, 292, 293
generator program, 290
series, testing of, 290–294
Runge-Kutta method, 147–154
application of, 149–154
program, 150
subroutine for, 151

—S—

Scalar matrix multiplication, 40
Sequential access files, 245–254
reading and writing, 246–250
Sequential file management program, 250–254
Simplex procedure,
linear programming and, 165–177
program, 166–177
Simulation, next step, 299–304
program, 301–303
Sorting, 233–241
bubble, 233–234
program, 234
quicksort, 235–239
program, 241
Stack subroutine, 271
Stacks, 270–272
Statistics,
measures of central tendency, 12–15
mean, 12–13
median, 13–14
mode, 14–15
measures of dispersion, 15–17
range, 15–16
standard deviation, 16–17
variance, 16–17
see also Data reduction
Stepwise regression, 58–104
program, 59–70
options of, 71–72
using the, 72–104
Subroutines,
bidirectional linked lists, 282–283
chi-square goodness of fit, 292
frequency plot, 22–25
integration trapezoidal rule, 139
joint observations plot, 26–27
linked lists, 278–279
lower triangular matrix, 48
matrices, 38–40
matrix addition, 39
matrix dimensions, 36
matrix exponentiation, 43
matrix multiplication, 41
using, 42
matrix transpose, 37
Newton-Raphson method, 128–130
plotting of, 18–32
random number generator, 290
Runge-Kutta method, 151
varying step size, 154
scalar matrix multiplication, 40
scaling, 30
solving simultaneous linear equations,
Gauss-Jordon method, 115–116
Gauss-Seidel method, 120–121
matrix inverse, 110, 111
sorting, 28–29
stack, 271
transpose, using the, 37
upper triangle matrix, 45
using, 46
see also Programs

—T—

Transpose of a matrix, 36–38
Trapezoidal rule of numerical integration, 138–141

—U—

Uniform distribution, 294–295
generator, 295

—V—

Variable names, 7
Vectors, 49

Documentation for the Program Diskette

About the Diskette

This diskette contains the subroutines and stand-alone programs which are presented and discussed in *BASIC Engineering, Science and Business Programs for the Apple II and IIe*. There is some specific information about the diskette which should be useful to you.

The diskette is almost full. It holds 38 major subroutines and programs and a host of supporting subroutines. You must write the main programs to utilize the subroutines. Main programs, which you can use, are presented in the book.

The subroutines have unique line numbers. This allows you to have more than one subroutine in memory at one time. To use a subroutine(s) with a main program you have entered, you can MERGE it by performing several steps. First SAVE your main program on a diskette. You can, of course, save it under any name you wish; we use the name MAIN, here.

```
]SAVE MAIN
```

Load the System Master diskette in the default drive and run the RENUMBER program.

```
]RUN RENUMBER
```

This step allows you to protect a program in a HOLD file while loading another program into memory. You now want to load the program MAIN back into memory.

```
]LOAD MAIN
```

In order to protect the main program when the subroutine is loaded, you can place it in HOLD. The following command accomplishes this:

```
]&H
```

The computer will tell you that the program is in HOLD. Now load the subroutine you wish to use (we call this SUB, here).

```
]LOAD SUB
```

Finally, you can merge the main program to the subroutine by removing it from the HOLD file and placing it in memory. Simply type:

```
]&M
```

You want to be sure that no line numbers in SUB also appear in MAIN. If SUB line numbers do appear in MAIN, you will have duplicate line numbers after the programs are merged.

One of the first things you will want to do is prepare a backup copy of the programs on another diskette to prevent their inadvertant loss.

Subroutine and Program Index

Below is a summary of the contents of each file on the diskette. The format used to describe each file is:

FILENAME, designation of program type (program or subroutine), reference to chapter in text where a detailed explanation can be formed, and a short description of what the program does.

CONTENTS, program, No chapter reference

This program is a handy reference to the programs and subroutines of this book. It provides a brief explanation of the subroutines and programs, including chapter and line numbers. Using this program you may access information describing a specific subroutine.

ANPLOT, subroutines, CHAPTER 2

Subroutines are included here for data anlysis and plotting. Sample statistics (mean, median, mode, range, standard deviation, variance) are calculated in one subroutine. Other subroutines generate frequency histograms and joint plots. Sorting and scaling subroutines are also included.

MATOP, subroutines, CHAPTER 3

This series of subroutines can be used for matrix operations. Matrix addition, multiplication, exponentiation, determinant, and inverse are included. Many auxiliary subroutines are used to assist in these operations.

SIMREG, program, CHAPTER 4

This program will estimate a line $Y = A + BX$ using simple linear regression. X is the independent variable and Y is the dependent variable.

STEPWISE, program, CHAPTER 4

This program will perform multiple or stepwise linear regression. Some data transforms are provided. Data management routines are also included. A forecast can be made using the estimated regression equation.

SIMLQ, subroutines, CHAPTER 5

This chapter presents a series of subroutines to solve simultaneous linear equations. Two analytic techniques, the matrix inverse and Gauss-Jordan methods, are presented as well as a numerical technique, Gauss-Seidel.

NEWRAP, subroutines, CHAPTER 6

This subroutine presents the Newton-Raphson procedure for locating the roots of a polynomial.

NUMINT, subroutines, CHAPTER 7

The subroutines listed here are used to perform numerical integration. Two different techniques are presented, integration by the trapezoidal rule and Legendre-Gauss quadrature.

DIFEQ, subroutines, CHAPTER 8

Two subroutines are presented for the numerical solution to ordinary first-order differential equations. The Runge-Kutta method is used. One subroutine uses a fixed step size while the other uses a variable step size.

SIMPLEX, program, CHAPTER 9

This program uses the simplex algorithm to solve linear programming problems. An editing subroutine is included to assist in data input.

EXSMOOTH, program, CHAPTER 10

This program will perform time series forecasting using Winter's method for exponential smoothing. The time series can contain trend and seasonal factors. Data management subroutines are provided.

CPM, program, CHAPTER 11

This program uses the critical path method (CPM) to determine the estimated project completion time and the associated critical path. Data management subroutines are also provided.

BUBSORT, subroutine, CHAPTER 12

This subroutine will sort a series of numbers into increasing or decreasing order using the bubble sort algorithm.

QUICKSORT, subroutine, CHAPTER 12

This subroutine will sort a series of numbers into increasing or decreasing order using the quicksort algorithm. In general, this algorithm is faster than the bubble sort algorithm. One exception is when the series of numbers is initially "almost" in the proper sequence.

SEQMAN, program, CHAPTER 13

This program illustrates the use of the sequential access method to manage data. A menu is provided that lets the user interactively add, delete, modify, and list the data using a sequential access disk file.

RANMAN, program, CHAPTER 13

This program illustrates the use of the random access method to manage data. A menu is provided that lets the user interactively add, delete, modify, and list data using a random access disk file.

STACK, subroutines, CHAPTER 14

These subroutines can be used to implement a stack data structure. Subroutines are provided that will initialize a stack and perform additions and deletions to a stack.

QUEUE, subroutines, CHAPTER 14

These subroutines can be used to implement a queue data structure. Subroutines are provided that will initialize a queue and perform additions and deletions to a queue.

LLQUEUE, subroutines, CHAPTER 14

These subroutines illustrate the use of a linked list. Each list is assumed to be a queue data structure. Each data element is assumed to be an integer. Subroutines are provided for initialization, addition and deletion using linked lists (queues).

BIDILIST, subroutines, CHAPTER 14

These subroutines illustrate the use of a bidirectional linked list. Each list consists of a set of alphanumeric strings which are maintained in alphabetical order. Subroutines are provided that will initialize, and perform additions and deletions to bidirectional linked lists.

ORGRND, subroutine, CHAPTER 15

This subroutine will generate uniformly distributed random numbers in the range of $0 \leq X \leq 1.0$, using the congruential method. The user can specify the initial random number seed.

CHISQ, subroutine, CHAPTER 15

This subroutine can be used to perform a chi-square goodness of fit test using a series of numbers in the range of $0 \leq X < 1.0$. This test can be used to determine if the hypothesis can be accepted that a series of numbers is not uniformly distributed.

DISTGEN, subroutines, CHAPTER 15

A set of subroutines is provided that will generate random numbers from any of the following distributions: uniform, normal, Poisson, exponential, and Erlang. Parameters necessary to specify a particular distribution, such as mean and variance, must be provided.

QUEUESIM, program, CHAPTER 15

This program illustrates next event simulation. A single-server queueing system is simulated. The time between arrivals to the system are exponentially distributed and the service times are normally distributed.

BASIC ENGINEERING, SCIENCE, AND BUSINESS PROGRAMS FOR THE APPLE II AND IIe

Philip M. Wolfe and C. Patrick Koelling

Now—a book that gives engineers, scientists, and business-oriented users alike a wide range of practical, ready-to-use programs for on-the-job or classroom use! From data manipulation to problem-solving methods, this unique "tool kit" text offers important computer techniques through the extensive use of ready-made programs and examples designed to help you get the most from your Apple II and IIe. It's so self-contained and easy-to-use, all you have to do is select the application you need, load your disk, key it in, and you're on your way to fast effective problem-solving!

In this book you will find these practical techniques:

- Data Analysis
- Plotting
- Matrix Operations
- Curve Fitting
- Solving Simultaneous Equations
- Roots of Polynomials
- Numerical Integration
- Numerical Solution to Differential Equations
- Linear Programming
- Forecasting & Programming
- Data Management Using Sequential & Random Access Disk Files
- Random Numbers & Simulations

CONTENTS

The Apple IIe/Data Reduction/Matrices & Vectors/Curve Fitting with Linear Regression/Solution of Simultaneous Linear Equations/Roots of Polynomials/Numerical Solutions to Differential Equations/Linear Programming/Forecasting with Exponential Smoothing/Project Planning & Scheduling with CPM/Sorting/Disk Data Files/Data Structures/Random Numbers & Simulations

ALSO AVAILABLE. . .OPTIONAL DISKETTE

This diskette includes the major programs from the text! See insert inside for ordering information.

ISBN 0-89303-284-0

Wolfe
Koelling

BASIC Engineering, Science, and Business Programs for the Apple II and IIe

Brady